

A11102 679009

NAT'L INST OF STANDARDS & TECH R.I.C.



A11102679009

Weppner, Matthew B/Image processing for  
QC100 .U56 NO.87-3065 1987 V19 C.1 NBS-P

NBS

PUBLICATIONS

REFERENCE

-3065

# IMAGE PROCESSING FOR OPTICAL ENGINEERING APPLICATIONS

---

---

Matthew B. Weppner  
Matt Young

National Bureau of Standards  
U.S. Department of Commerce  
Boulder, Colorado 80303-3328

April 1987

QC

100

.U56

#87-3065

1987



NBS  
20100  
.U56  
no. 87-3065  
1987  
C.2

NBSIR 87-3065

# IMAGE PROCESSING FOR OPTICAL ENGINEERING APPLICATIONS

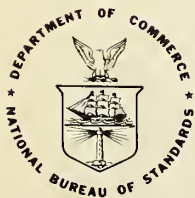
---

---

Matthew B. Weppner  
Matt Young

Electromagnetic Technology Division  
Center for Electronics and Electrical Engineering  
National Engineering Laboratory  
National Bureau of Standards  
Boulder, Colorado 80303-3328

April 1987



---

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, Secretary

NATIONAL BUREAU OF STANDARDS, Ernest Ambler, Director



## Contents

	Page
1. Introduction.....	1
1.1 Statement of Purpose and Organization.....	1
1.2 Image Representation and Format.....	2
2. Image Processing Functions.....	3
2.1 Single-Pixel Functions.....	3
2.1.1 Biasing.....	5
2.1.2 Contrast Enhancement.....	5
2.1.3 Negative Formation.....	7
2.1.4 Thresholding.....	9
2.2 Multiple-Pixel Functions.....	10
2.2.1 Convolution.....	10
2.2.2 Fourier Transforms.....	13
2.2.3 Magnification/Demagnification.....	21
2.2.4 Noise Filtering.....	23
2.2.5 Resolution Changing.....	25
2.3 Two-Picture Functions.....	30
2.3.1 Averaging.....	30
2.4 Image Information Display Tools.....	33
2.4.1 Gaussian Curve Fitting.....	33
2.4.2 Multimode Curve Fitting (g-Profile).....	35
2.4.3 Intensity Histogram.....	37
2.4.4 Line Intensity Scan.....	40
2.4.5 Three-Dimensional Intensity Plot.....	40
3. Applications of the Software.....	40
3.1 Creating a Model Image for a Hybrid Computer- Optical Processing Experiment.....	40
3.2 Analyzing Near- and Far-Field Images for an Integrated Optics Experiment.....	43
4. Conclusion.....	52
4.1 Conclusions and Future Applications.....	52
5. References.....	54
Appendix A. User's Manual.....	A-1
1. Introduction.....	A-1
1.1 Software Description.....	A-1
2. Program Information.....	A-1
2.1 Picture File Specifications.....	A-1
2.2 Program Execution.....	A-2
2.3 Program Compilation.....	A-3
3. Commands.....	A-3
3.1 Image Commands.....	A-3
3.1.1 Average.....	A-3
3.1.2 Bias.....	A-4
3.1.3 Convolution.....	A-4
3.1.4 Demagnify.....	A-5
3.1.5 Enhance Contrast.....	A-5
3.1.6 Fourier Transform.....	A-5
3.1.7 Magnify.....	A-6
3.1.8 Negative.....	A-6
3.1.9 Pratt's Noise Cleaning Algorithm.....	A-6
3.1.10 Resolution Change.....	A-7
3.1.11 Threshold.....	A-7

3.2 Display Commands.....	A-7
3.2.1 Gaussian Curve Fit.....	A-7
3.2.2 Multimode Curve Fit (g-Profile).....	A-8
3.2.3 Intensity Histogram.....	A-8
3.2.4 Line Intensity Scan (2-D).....	A-8
3.2.5 3-D Intensity Plot.....	A-9
3.3 File Commands.....	A-9
3.3.1 Read.....	A-9
3.3.2 Write.....	A-9
Appendix B. Source Code.....	B-1
Appendix C. Hybrid Computer-Optical Processing with Inexpensive Liquid Crystal Television.....	C-1
Abstract.....	C-1
Introduction.....	C-1
Hybrid optical processor.....	C-1
Experiment.....	C-4
Results.....	C-5
Discussion.....	C-7
Acknowledgments.....	C-8
References.....	C-8

# Image Processing for Optical Engineering Applications

Matthew B. Weppner and Matt Young

Electromagnetic Technology Division  
National Bureau of Standards  
Boulder, Colorado 80303

This Internal Report describes a palette of computer image processing algorithms for picture processing and optical fiber characterization.

Keywords: computer processing; Fourier optics; g-profile; Gaussian beam; image processing; images; optical fiber characterization; optical processing; single-mode fiber

## 1. Introduction

### 1.1 Statement of Purpose and Organization

This Internal Report describes the development and testing of image processing software designed for optical engineering applications. Image processing functions in this software include two-dimensional Fourier transforms, convolution, noise reduction, multiple image resolutions, and low-level image processing functions. The software also contains image information display tools including Gaussian beam and g-profile characterization for optical fiber measurements. The necessary image file input/output routines are presented in the software and are used to read and store images in conjunction with other image processing software, digitizing cameras, and output display devices.

A design is never finished but simply stopped due to time limitations. Therefore, this software does not comprise a complete set of image processing functions and display tools, but, rather, provides a good base for most applications and can be expanded with additional future image processing algorithms.

The rest of this section describes the basic two-dimensional digitized image used in image processing. Section 2 describes in depth the image processing functions and display tools contained in the software. Section 3.1 describes an application of the software for a hybrid computer-optical image recognition experiment (a copy of the full paper is in appendix C). Section 3.2 describes a preliminary experiment that relates the near-field and far-field scans from a single-mode optical fiber. Section 4 contains concluding remarks and future applications. Appendices A and B show the manual and source code listing for the software.



Figure 1-1. American Gothic by Grant Wood (1930), oil on beaver board (29-7/8" x 25"). Friends of the American Art Collection. Copyright, The Art Institute of Chicago. All rights reserved.

Figure 1-1 is an image used throughout section 2. This painting was chosen because it is a familiar image with good range of contrast, shapes, and detail.

This report is based entirely on the Master of Science thesis of Matthew B. Weppner (Electrical and Computer Engineering Department, University of Colorado, Boulder, 1986). It has been edited lightly, and the format has been changed, but it is otherwise intact.

## 1.2 Image Representation and Format

A digitized image is made up of picture elements called pixels. A pixel is a small region, usually square, whose size determines the resolution of the picture. Each pixel is assigned a numerical value which represents the average light intensity over the area of the pixel. The intensity values are not absolute measurements but represent a relative intensity level. In this software, each pixel value must be a positive integer, one byte, which allows for

256 intensity levels. Possible pixel values are 0 to 255, and the maximum dynamic range over the 256 intensity levels is

$$\text{maximum dynamic range} = 10 \cdot \log_{10} \left( \frac{255}{1} \right) \approx 24 \text{ dB.} \quad (1)$$

Although this represents the maximum dynamic range of the software, the dynamic range of a digitized image is determined by the weakest link among the input digitizing device, software, and output device.

Each digitized two-dimensional image is made up of a finite number of pixels with a horizontal width and vertical height (resolution format) specified by the software. The image is represented by a two-dimensional array,  $p(x,y)$ , which is shown in figure 1-2. Each element  $p(x_0,y_0)$  represents the intensity value of a pixel at a specified position  $(x_0,y_0)$ . In this software valid resolution formats are 256 x 240, 128 x 120, 64 x 60, 32 x 30, and 16 x 15.

Figure 1-3 is a 256 x 240 digitized image of figure 1-1. A CCD (charge coupled device) camera with resolution 480 x 384 is used to obtain this image. Four similar pictures have been averaged together to reduce the effects of video camera noise (section 2.3).

## 2. Image Processing Functions

### 2.1 Single-Pixel Functions

Single-pixel functions are those in which the (intensity) value of a pixel is altered according to an algorithm that depends on the original value of that pixel. The functions have the general form,

$$p(x_0,y_0) = f_{\text{single}}[p(x_0,y_0)], \quad (2)$$

where  $p(x_0,y_0)$  is the current pixel being processed in the picture  $p(x,y)$ .

These functions scan the entire image and process each pixel individually. In general these functions are relatively quick to execute and do not require complex computation.

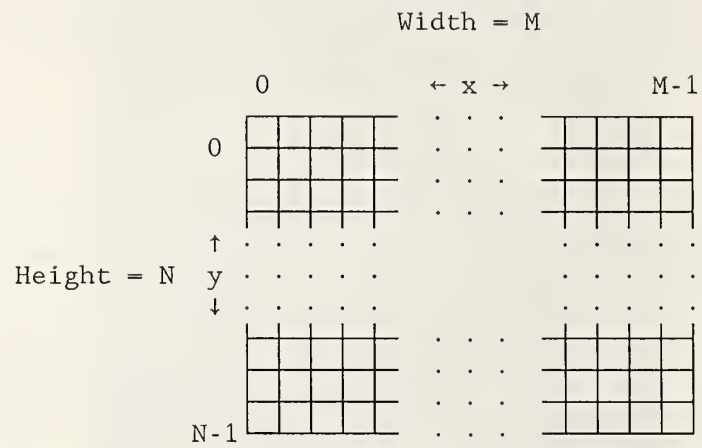


Figure 1-2. Diagram of a  $M \times N$  digitized image  $p(x,y)$ .



Figure 1-3. 256x240 digitized image of figure 1-1.

### 2.1.1 Biasing

Biasing adds an input scalar value to each pixel of a digitized image. The input value must be an integer within the valid range (0 to 255). The general form of the equation is

$$f_{\text{bias}}(p(x_0, y_0), \beta) = \begin{cases} 0 & p(x_0, y_0) + \beta < 0 \\ p(x_0, y_0) + \beta & 0 \leq p(x_0, y_0) + \beta \leq 255 \\ 255 & p(x_0, y_0) + \beta > 255 \end{cases} \quad (3)$$

where  $\beta$  = input bias value and has the range  $-255 < \beta < 255$ .

The purpose of this biasing function is to shift the spectrum of intensity levels by a specified amount. This changes the average intensity level of the picture and can alter the range of visual contrast of an image. Image information loss results if computed values exceed the intensity range and are truncated.

Figure 2-1 shows the result of biasing figure 1-3 with  $\beta = 50$ . Figure 1-3 initially had a good range of exposure. In figure 2-1 the picture appears much brighter, but the brightest parts of the picture appear overexposed because the pixel values in those areas exceed 255. Figure 2-2 shows the result of biasing figure 1-3 with  $\beta = -50$ . In this case the picture is much darker and much of the dark area appears underexposed because the pixel values in those areas have values that are less than 0.

### 2.1.2 Contrast Enhancement

Contrast enhancement increases the range of contrast of the intensity values of a picture. Pixel values within an input range less than 256, are linearly expanded to the full range of 0 to 255. The input range is determined by a lowest and highest input bounds. The general form of the equation is

$$f_{\text{enhance}}(p(x_0, y_0), a, b) = \begin{cases} 0 & \alpha < 0 \\ \alpha & 0 \leq \alpha \leq 255 \\ 255 & \alpha > 255 \end{cases} \quad (4)$$



Figure 2-1. Biased image of figure 1-3 ( $=+50$ ).



Figure 2-2. Biased image of figure 1-3 ( $= -50$ ).

where  $\alpha = \left(\frac{255}{b-a}\right) (p(x_0, y_0) - a)$ ,

a = input lower bound,

b = input upper bound,

$0 < a < b < 255$ .

Pixel intensity values not within the range of the lowest and highest input values will exceed the valid range after enhancement and are truncated to 0 and 255, respectively.

Contrast enhancement makes the picture appear brighter and brings out hidden detail. This function is also useful for expanding the dynamic range of intensities of a picture for other image processing functions.

Figure 2-3 shows an enhanced image of Figure 1-3 using the values  $a = 49$  and  $b = 227$ , which are the lowest and highest pixel values in figure 1-3. We call this process "auto-enhance" in the software because the image is automatically adjusted to the maximum possible contrast. Figure 2-4 shows another enhanced image of Figure 1-3 using  $a=100$  and  $b=175$ . This image is deliberately over-enhanced to bring out the detail within range between  $a$  and  $b$ . Pixel information outside this range is lost.

### 2.1.3 Negative Formation

Forming a negative of a digitized picture is analogous to creating negative image in photography. The range of pixel intensities is reversed, with the brightest pixels becoming the darkest and vice versa. The form of the equation is

$$f_{\text{negative}}(p(x_0, y_0)) = 255 - p(x_0, y_0). \quad (5)$$

The purpose of this function is to reverse the contrast of a picture and create an inverse image. Figure 2-5 shows the negative image of figure 1-3. This function is helpful in bringing out detail and analyzing certain images.



Figure 2-3. Enhanced image of figure 1-3 ( $a=49$ ,  $b=227$ )



Figure 2-4. Enhanced image of figure 1-3 ( $a=100$ ,  $b=175$ ).



Figure 2-5. Negative image of figure 1-3.

#### 2.1.4 Thresholding

Thresholding is used to set a range of pixels to a desired value. Upper and lower pixel input bounds are used to establish a range of intensity. The general form of the function is

$$f_{\text{threshold}}(p(x_0, y_0), a, b, \tau) = \begin{cases} \tau & a \leq p(x_0, y_0) \leq b \\ p(x_0, y_0) & \text{otherwise} \end{cases} \quad (6)$$

where  $a$  = input lower bound,  
 $b$  = input upper bound,  
 $0 \leq a < b \leq 255$ ,  
 $\tau$  = input threshold set value,  
 $0 \leq \tau \leq 255$ .

This function sets a range of pixel intensity values equal to an input threshold value. This can be used to distinguish ranges of intensity within a picture.

Figure 2-6 shows the thresholded image of figure 1-3 with the values  $a = 0$ ,  $b = 128$ , and  $\tau = 64$ . Contrast information within this range, 0 to 128, is lost and diverts our attention to the unthresholded pixel values. Figure 2-7 shows an image of figure 1-3 with the number of possible pixel intensity values reduced from 256 to 16. This is done by setting ranges of pixels into 16 threshold levels, by thresholding 16 times using  $a = 0$ ,  $b = 15$ , and  $\tau = 0$ , then  $a = 16$ ,  $b = 31$ , and  $\tau = 16$ , etc.

## 2.2 Multiple-Pixel Functions

Multiple-pixel functions are image processing functions which operate on one pixel as a function of many pixels or even the entire digitized image. The functions have the general form,

$$p'(x_0, y_0) = f_{\text{multiple}}(x_0, y_0, p(x, y)), \quad (7)$$

where  $p'(x_0, y_0)$  is the current pixel being processed in the picture  $p(x, y)$ .

These functions scan the entire picture but process each pixel individually. Since each pixel is a function of many pixels (or the entire image) we must put the processed results into a second array,  $p'(x, y)$ . This is necessary because the computer is capable of processing only one pixel at a time, and the entire original image is needed until all the pixels are processed. Usually, these functions are relatively slow to execute and require more complex computation compared to single pixel functions.

### 2.2.1 Convolution

The purpose of the convolution function is to apply a spatial frequency filter to the digitized picture. A straightforward method [1] of convolving two functions is to multiply the Fourier transform of the digitized image with a frequency filter function in the frequency plane. The inverse Fourier transform is then taken to transform the result back into the image plane. The general form is

$$p'(x, y) = F^{-1}\{P(f_x, f_y) \cdot K(f_x, f_y)\}, \quad (8)$$



Figure 2-6. Thresholded image of figure 1-3 ( $a=0$ ,  $b=128$ ,  $\tau=64$ ).



Figure 2-7. Thresholded image of figure 1-3 with the number of pixel intensity values reduced from 256 to 16.

where  $F\{g(x,y)\} = G(f_x, f_y)$  = (forward) Fourier transform,  
 $F^{-1}\{G(x,y)\} = g(x,y)$  = inverse Fourier transform,  
 $P(f_x, f_y) = F\{p(x,y)\}$  = Fourier transform of the picture,  
 $K(f_x, f_y)$  = frequency filter function.

Computing the two-dimensional Fourier transform of the picture and the inverse transform of the filtered result is usually very time consuming. We can also filter the picture with respect to spatial frequency by performing the convolution directly in the image plane. Using the convolution theorem we obtain [1],

$$F\{k(x,y) * p(x,y)\} = P(f_x, f_y) \cdot K(f_x, f_y),$$

$$k(x,y) * p(x,y) = F^{-1}\{P(f_x, f_y) \cdot K(f_x, f_y)\} = p'(x,y), \quad (9)$$

where  $f * g = \int_{-\infty}^{+\infty} \int_{-\infty}^{-\infty} g(\xi, \eta) g(x-\xi, y-\eta) d\xi d\eta,$

$k(x,y) = F^{-1}\{K(f_x, f_y)\}$  = convolution kernel function,  
and  $*$  denotes convolution. Convolution in the image plane is the mathematical equivalent to multiplication in the Fourier plane. We use convolution because it is simpler to implement and faster when the kernels are small.

In image processing we deal with digitized pictures in matrix form. Similarly, we must chose a finite matrix,  $k(x,y)$ , which serves as the convolution kernel. This convolution kernel is often an approximation to the inverse Fourier transform of the frequency filter function,  $K(f_x, f_y)$ . The general form of the two-dimensional computer convolution formula is

$$f_{\text{convolve}}(x_o, y_o, p(x,y), k(x,y)) = \begin{cases} 0 & \alpha < 0 \\ 255 & \alpha > 255 \\ \alpha & \text{otherwise} \end{cases}$$

where

$$\alpha = \beta + \frac{1}{\eta} \cdot \left( \sum_{n=1}^{N_k} \sum_{m=1}^{M_k} k(m,n) \cdot p(x_o+m-m_c, y_o+n-n_c) \right), \quad (10)$$

$\beta$  = bias value,

$\eta$  = normalization value,

$$n = \begin{cases} 1 & \sum_{n=1}^{N_k} \sum_{m=1}^{M_k} k(m,n) = 0 \\ \sum_{n=1}^{N_k} \sum_{m=1}^{M_k} k(m,n) & \text{otherwise} \end{cases}$$

$k(x,y) = M_k \times N_k$  matrix (convolution kernel),

$(m_c, n_c)$  = center of matrix  $k(x,y)$ ,

$m_c = (M_k \text{ div } 2) + 1$ ,

$n_c = (N_k \text{ div } 2) + 1$ ,

and "div" denotes integer division, which ignores the remainder.

Figure 2-8 shows a low-pass image of figure 1-3. The convolution kernel is

$$k_{\text{low-pass}} = \begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix}. \quad (11)$$

This kernel is a rough approximation to a small circles in the Fourier plane. The result is loss of high frequency information or blurring.

Figure 2-9 shows a high-pass filtered image of figure 1-3. The convolution kernel is

$$k_{\text{high-pass}} = \begin{vmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{vmatrix}. \quad (12)$$

This kernel is a rough approximation to a sinc(r) function, which represents an opaque stop in the Fourier plane. The result is loss of low frequency information and gives an image with only the edges and noise remaining.

## 2.2.2 Fourier Transforms

The Fourier transform functions transform a digitized image in either the forward or reverse direction. This is similar to the Fourier transforming



Figure 2-8. Low-pass filtered image of figure 1-3.



Figure 2-9. High-pass filtered image of figure 1-3.

properties of a lens, although a lens performs only a forward transformation. The form of the transform functions is

$$P'(f_x, f_y) = f_{\text{forward-FFT}}(P(x, y), \beta, \epsilon), \quad (13)$$

$$p'(x, y) = f_{\text{inverse-FFT}}(P(f_x, f_y), \beta, \epsilon), \quad (14)$$

where  $P(f_x, f_y)$  = the 2-D Fourier transform of  $p(x, y)$ ,  
 $x, y$  = spatial variables in the image plane,  
 $f_x, f_y$  = frequency variables in the Fourier plane  
 $\beta$  = transform plane magnification factor,  
 $\epsilon$  = transform plane scaling factor.

In this software a digitized image is assumed to be a phaseless intensity distribution. To process the Fourier transform we convert the intensity distribution into a complex amplitude distribution. For the forward transform,

$$\begin{aligned} p(x_o, y_o) &= |a_{\text{complex}}(x_o, y_o)|^2 \\ &= (a_{\text{real}}(x_o, y_o))^2 + (a_{\text{imaginary}}(x_o, y_o))^2 \end{aligned} \quad (15)$$

where  $a_{\text{complex}}(x, y)$  = complex amplitude distribution in the image plane. Arbitrarily, we set

$$\begin{aligned} a_{\text{real}}(x_o, y_o) &= (p(x_o, y_o))^{1/2}, \\ a_{\text{imaginary}}(x_o, y_o) &= 0. \end{aligned} \quad (16)$$

Similarly for the inverse transform,

$$\begin{aligned} P(f_{x_o}, f_{y_o}) &= |A_{\text{complex}}(f_{x_o}, f_{y_o})|^2 \\ &= (A_{\text{real}}(f_{x_o}, f_{y_o}))^2 + (A_{\text{imaginary}}(f_{x_o}, f_{y_o}))^2, \end{aligned} \quad (17)$$

where  $A_{\text{complex}}(f_x, f_y)$  = complex amplitude distribution in the Fourier plane. Again, we set

$$A_{\text{real}}(f_{x_0}, f_{y_0}) = (P(f_{x_0}, f_{y_0}))^{1/2}$$

$$A_{\text{imaginary}}(f_{x_0}, f_{y_0}) = 0. \quad (18)$$

To describe the transforming process in detail we start with the basic Fourier transform equations. The equation for the forward Fourier transform is [1]

$$F\{g(x,y)\} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} g(x,y) \cdot e^{-j2\pi(f_x x + f_y y)} dx dy = G(f_x, f_y), \quad (19)$$

and the inverse Fourier transform is

$$F^{-1}\{G(f_x, f_y)\} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} G(f_x, f_y) \cdot e^{+j2\pi(f_x x + f_y y)} df_x df_y = g(x,y). \quad (20)$$

Since we are interested in performing the two-dimensional Fourier transform on a digitized picture, we use the Fast Fourier Transform algorithm (FFT) to take advantage of its inherent speed, which is necessary for the transformation of the relatively large digitized pictures. The FFT algorithm in its basic form transforms only a one-dimensional array. To perform the Fourier transform in two dimensions we rewrite the two-dimensional transform as successive one-dimensional transforms:

$$\begin{aligned} F\{g(x,y)\} &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} g(x,y) \cdot e^{-2\pi f_x x} \cdot e^{-j2\pi f_y y} dx dy \\ &= \int_{-\infty}^{+\infty} \left( \int_{-\infty}^{+\infty} g(x,y) \cdot e^{-2\pi f_x x} dx \right) \cdot e^{-2\pi f_y y} dy \\ &= F_y\{F_x\{g(x,y)\}\} = F_x\{F_y\{g(x,y)\}\}. \end{aligned} \quad (21)$$

Similarly for the inverse transform,

$$F^{-1}\{g(f_x, f_y)\} = F_y^{-1}\{F_x^{-1}\{g(f_x, f_y)\}\} = F_x^{-1}\{F_y^{-1}\{g(f_x, f_y)\}\}. \quad (22)$$

Thus, to obtain the two-dimensional transforms we can first transform all the (vertical) columns, and then transform all the (horizontal) rows, or vice versa.

If the input picture function has moderate size relative to the bounds of the picture array, the transformed result will occupy a small portion of the transform plane and will not contain many nonzero data points or pixels. To achieve a transformed result of moderate size with an input of moderate size we use transform plane magnification. This results in a transform with more nonzero elements, though there is, in reality, no more information than before. Its usefulness is mainly for visual display.

Transform plane magnification involves shrinking the input plane to achieve a stretching or magnification in the transformed plane. Using the similarity theorem for Fourier transforms we see that [1]

$$F\{g(ax,by)\} = \frac{1}{|a \cdot b|} \cdot G\left(\frac{f_x}{a}, \frac{f_y}{b}\right). \quad (23)$$

Thus, for the magnified forward and inverse transforms,

$$F\left\{p\left(\frac{x}{\beta}, \frac{y}{\beta}\right)\right\} = \beta^2 \cdot P(\beta f_x, \beta f_y), \quad (24)$$

$$F^{-1}\left\{P\left(\frac{f_x}{\beta}, \frac{f_y}{\beta}\right)\right\} = \beta^2 \cdot p(\beta x, \beta y), \quad (25)$$

where  $\beta$  = transform plane magnification factor. To obtain a shrinking of the input plane without loss of information, the input plane is enlarged by a factor of  $\beta$  and padded with 0's. This creates a factor-of- $\beta$  magnification in the transform plane with no loss of information in the input plane because the input is only effectively shrunk with the use of larger FFT arrays. The penalty is increased computing time.

One of the requirements of the FFT algorithm is that the number of elements in the one-dimensional input array be a power of 2. Therefore, for normal situations ( $\beta = 1$ ), the number of elements in each one-dimensional FFT array must also be a power of 2. In both the horizontal and vertical

directions, the size of the FFT array must equal the number of pixels in that direction. If the number of pixels is not a power of 2, the number of elements used for the FFT array must be the next higher power of 2. For an M x N picture,

$$a = \begin{cases} \max_{\text{int}}(\frac{\log M}{\log 2}) & \text{for horizontal transforms,} \\ \max_{\text{int}}(\frac{\log N}{\log 2}) & \text{for vertical transforms,} \end{cases} \quad (26)$$

where  $2^a$  = minimum number of elements in the FFT array, and  
 $\max_{\text{int}}()$  = rounds up to the next integer.

To adjust for the Fourier plane magnification, we use

$$A = \beta \cdot 2^a \quad (27)$$

where A = number of elements in the FFT array to be used, and  
 $\beta = 2^n$  for  $n \geq 0$ .

Since a is an integer,  $\beta$  must be a power of 2 to ensure that A is also a power of 2.

Since the number of elements used in either direction usually exceeds the numbers of pixels, extra FFT array elements are set to 0 before being transformed. After transformation, the extra FFT array elements are ignored.

We are careful in choosing an input image that contains 0's around the boundaries of the two-dimensional picture array to prevent ringing due to discontinuous edges. Also, a convention set by the FFT algorithm requires that the zero frequency term (the dc term) be shifted to the first element of the array to be transformed. The transformed result also assumes that the first element is the dc term. Therefore, the horizontal rows and vertical columns are shifted to move the center dc term from the assumed center of the picture (appendix A) to the upper left position (0,0). After the transformation the rows and columns are shifted once again to move the dc term back to the center position.

Once the complex amplitude distribution is obtained the array is transformed. The vertical columns of the picture are transformed first and the horizontal rows are transformed second. For the forward transform,

$$A_{\text{complex}}(f_x, f_y) = \text{FFT}_x\{\text{FFT}_y\{a_{\text{complex}}(x, y)\}\}, \quad (28)$$

and for the inverse transform,

$$a_{\text{complex}}(x, y) = \text{FFT}_x^{-1}\{\text{FFT}_y^{-1}\{A_{\text{complex}}(f_x, f_y)\}\}. \quad (29)$$

Now the transformed amplitude distribution must be converted back into an intensity distribution. For the forward transform,

$$P(f_{x_0}, f_{y_0}) = (A_{\text{real}}(f_{x_0}, f_{y_0}))^2 + (A_{\text{imaginary}}(f_{x_0}, f_{y_0}))^2, \quad (30)$$

and for the inverse transform,

$$p(x_0, y_0) = (a_{\text{real}}(x_0, y_0))^2 + (a_{\text{imaginary}}(x_0, y_0))^2. \quad (31)$$

Since phase is not retained after converting back to an intensity distribution, these transformed images are not completely analogous to amplitude transforms, which have phase. Consequently, the Fourier transform functions described here are not conservative, and we cannot undo the transform process. Using the Fourier integral theorem for amplitude distributions [1] we see that

$$g(x, y) = F^{-1}\{F\{g(x, y)\}\} = F\{F^{-1}(g(x, y))\}. \quad (32)$$

But, since we are assuming intensity distributions with no retention of phase terms,

$$\begin{aligned} p(x, y) &\neq f_{\text{inverse-FFT}}(f_{\text{forward-FFT}}(p(x, y))) \\ &\neq f_{\text{forward-FFT}}(f_{\text{inverse-FFT}}(p(x, y))). \end{aligned} \quad (33)$$

We must be careful in interpreting these transform functions, keeping in mind that the digitized input pictures and transformed results are intensity distributions and have certain limitations.

Since Fourier transforms usually have large peaks we use an integer scaling factor,  $\epsilon$ , to enhance smaller details which would not normally be seen because of the limited dynamic range of the pixel values (0 to 255). This scaling process is similar to overexposing film to observe detail normally drowned out by the strong peaks. The scaling factor,  $\epsilon$ , represents the level of overexposure to which the transformed values are assigned pixel values. A scaling factor of 1 means the highest transformed FFT value is assigned 255 and the rest of the values are scaled accordingly. For values of  $\epsilon$  greater than 1 the scale is simply multiplied by that factor. Pixel values which are computed to be higher than 255 are truncated. For the forward transform,

$$P(f_{x_0}, f_{y_0}) = \begin{cases} \rho & 0 \leq \rho \leq 255 \\ 255 & \text{otherwise} \end{cases}$$

$$\rho = \left( \frac{255 \cdot \epsilon}{h} \right) \cdot P(f_{x_0}, f_{y_0}), \quad (34)$$

where  $\epsilon$  = integer scaling factor ( $\epsilon \geq 1$ ),  
 $h$  = highest value contained in  $P(f_x, f_y)$ .

For the inverse transform,

$$p(x_0, y_0) = \begin{cases} \rho & 0 \leq \rho \leq 255 \\ 255 & \text{otherwise} \end{cases}$$

$$\rho = \left( \frac{255 \cdot \epsilon}{h} \right) \cdot p(x_0, y_0), \quad (35)$$

where  $\epsilon$  = integer scaling factor ( $\epsilon \geq 1$ ),  
 $h$  = highest value contained in  $p(x, y)$ .

Figure 2-10 shows a 128 x 120 Fourier transform of figure 1-3. 128 x 120 is the highest resolution that can be transformed by our computer, since this resolution requires 120 kbytes of memory for the image alone using complex floating point arithmetic which requires 8 bytes per pixel (a 256 x 240 image requires 480 kbytes of memory). Since the image of figure 1-3 is large relative to the bounds of the image array, the resulting transform is quite small. Enlargement is achieved with a Fourier plane magnification factor  $\beta = 4$ . To bring out the finer transform detail the image was scaled or overexposed by a factor of  $\epsilon = 3000$ .

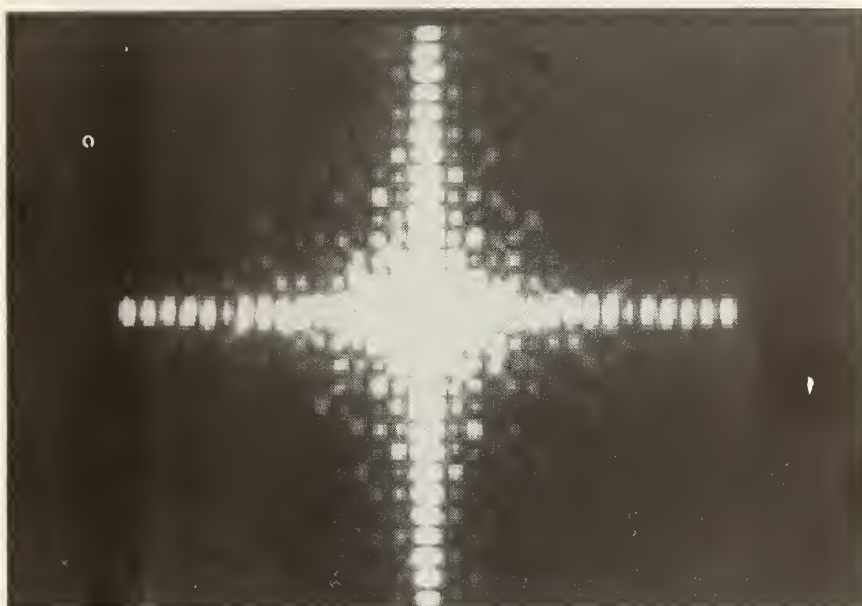


Figure 2-10. Fourier transform of figure 1-3.

Figure 2-11 shows a 128 x 120 image of a  $\text{circ}(r)$  function with  $r = 20$  and height = 255. Figure 2-12 shows the Fourier transform of figure 2-11, which has the form  $\text{sinc}^2(r)$ . Once again a Fourier plane magnification factor  $\beta = 4$  is used. To bring out the detail in the secondary maxima a scaling factor  $\epsilon = 50$  is used.

### 2.2.3 Magnification/Demagnification

Magnification increases the physical scaling of a digitized picture by a factor of 2 in both the horizontal and vertical directions. A factor of 2 is the only value provided by the software to ensure exact processing without aliasing between pixels. This process is similar to 2X magnification using lenses. Information contained in the picture is decreased by a factor of 4 because only the center fourth of the area of the image is capable of magnification. Information outside the area to be magnified is lost. Magnified pixels are simply replicated to produce the larger image. The general form of the equation is

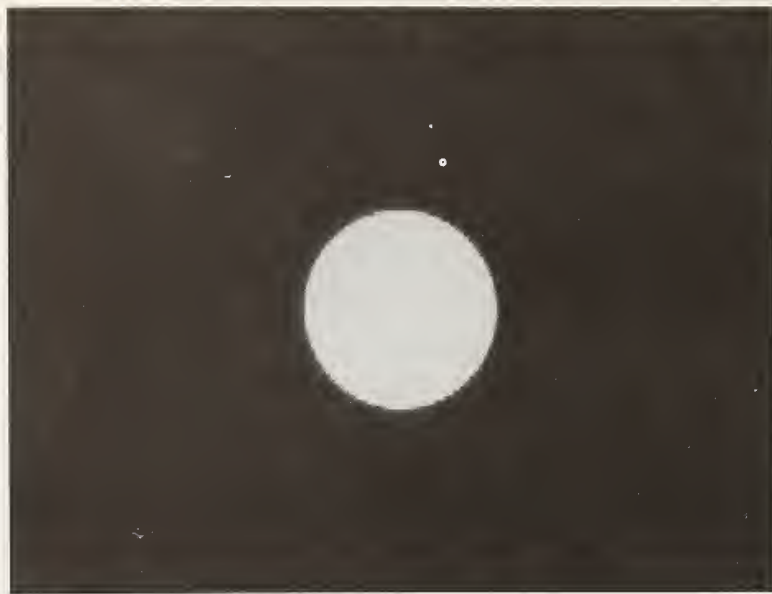


Figure 2-11. 128x120 computer generated  $\text{circ}(r)$  function with  $r=20$ .

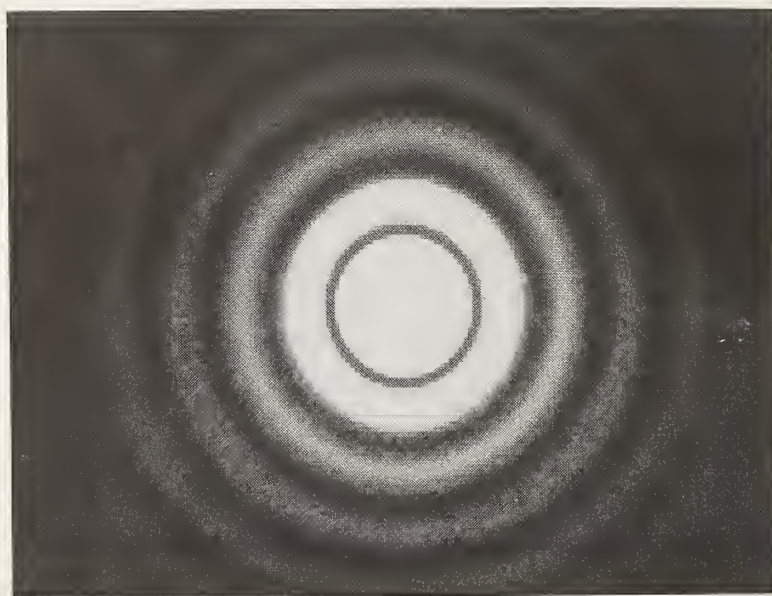


Figure 2-12. Fourier transform of figure 2-11.

$$f_{\text{magnify}}(x_0, y_0, p(x, y)) = p(x_a + (x_0 \text{ div } 2), y_a + (y_0 \text{ div } 2)) \quad (36)$$

where  $x_a = M \text{ div } 4$ ,  
 $y_a = N \text{ div } 4$ ,  
 $M \times N$  = picture resolution format.

Figure 2-13 shows the magnified image of figure 1-3.

Demagnification decreases the scaling of a digitized picture by a factor of 2 in both directions. Information in the picture is decreased by a factor of 4 because averaging of four pixels is used to create each demagnified pixel. This is similar to 2X image reduction using lenses. Information taken from beyond the bounds of the input image is obviously unavailable and the demagnified pixels are set to 0. The general form of the equation is

$$f_{\text{demagnify}}(x_0, y_0, p(x, y)) = \begin{cases} \alpha & x_a \leq x_0 \leq x_b, y_a \leq y_0 \leq y_b \\ 0 & \text{otherwise} \end{cases}$$

$$\alpha = \frac{1}{4} \sum_{j=1}^2 \sum_{i=1}^2 p(2(x_0 - x_a) + i, 2(y_0 - y_a) + j), \quad (37)$$

where  $x_a = M \text{ div } 4$      $x_b = 3(M \text{ div } 4)^{-1}$ ,  
 $y_a = N \text{ div } 4$      $y_b = 3(N \text{ div } 4)^{-1}$ ,  
 $M \times N$  = picture resolution format.

Figure 2-14 shows the demagnified image of figure 1-3. Both functions can be used repeatedly on an image to achieve higher magnification or demagnification factors.

#### 2.2.4 Noise Filtering

Noise filtering is used to reduce random noise such as video camera noise or noise generated by other image processing functions. The algorithm for the function is from reference [2]. When the intensity value of a pixel differs from the average of its eight surrounding neighborhood pixels by more than a set value,  $\epsilon$ , the pixels are set to the average neighborhood value. The general form of the function is



Figure 2-13. 2X magnified image of figure 1-3.



Figure 2-14. 2X demagnified image of figure 1-3.

$$f_{\text{noise-filtering}}(x_0, y_0, p(x, y), \epsilon) = \begin{cases} \frac{1}{8} \sum_{i=1}^8 p_i & \left| p(x_0, y_0) - \frac{1}{8} \sum_{i=1}^8 p_i \right| > \epsilon \\ p(x_0, y_0) & \text{otherwise} \end{cases} \quad (38)$$

where  $\epsilon$  = input set value,

$p_1, \dots, p_8$  = the eight neighboring pixels of  $p(x_0, y_0)$ .

The average of the neighboring pixels of this function is obtained by automatically convolving, in software, the image with the noise filtering convolution kernel,

$$k_{\text{noise-filtering}} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}. \quad (39)$$

The difference between  $p(x_0, y_0)$  and the average is then compared with  $\epsilon$ , and  $p(x_0, y_0)$  is set accordingly by the function.

Figure 2-15 is an image of figure 1-3 with randomly generated pixels of value 255 added. This random noise is similar to some types of video camera noise under extreme conditions. Figure 2-16 is an image of figure 2-15 with the noise filtering. Looking closely at the cleaned image we can see that the noise is greatly reduced but not completely eliminated.

### 2.2.5 Resolution Changing

The resolution changing function changes the horizontal-by-vertical resolution format of the input image. Changing resolution does not alter the scale of the picture but alters only the number of pixels in both the horizontal and vertical directions.

Decreasing resolution decreases the number of pixels by a factor of  $4^n$  by reducing the number of pixels in both directions by a factor of  $2^n$ . This results in a loss of information because groups of  $4^n$  pixels are averaged to produce pixels in the lower resolution picture. The general form of the equation is



Figure 2-15. Image of random noise superimposed on figure 1-3.



Figure 2-16. Reduced noise image of figure 2-15.

$$f_{\text{decrease-res}}(x_0, y_0, p(x, y), n) = \frac{1}{4^n} \sum_{j=1}^{2^n} \sum_{i=1}^{2^n} p(2^n \cdot x_0 + i, 2^n \cdot y_0 + j), \quad (40)$$

where  $n$  = resolution reduction factor ( $n > 1$ ). The new resolution format is

$$M' = \frac{M}{2^n} \text{ and } N' = \frac{N}{2^n}, \quad (41)$$

where  $M \times N$  = resolution format of  $p(x, y)$ ,

$M' \times N'$  = resolution format of  $p'(x, y)$ .

Figure 1-3 is the highest resolution format handled by the software at 256 x 240. Figures 2-17, 2-18, 2-19, and 2-20 are reduced resolution images of figure 1-3 with respective resolution formats, 128 x 120, 64 x 60, 32 x 30, and 16 x 15.

Increasing resolution increases the number of pixels by a factor of  $4^n$  by increasing the number of pixels in both directions by a factor of  $2^n$ . This results in no gain of information because pixels are simply replicated by a factor of  $4^n$ . The general form of the equation is

$$f_{\text{increase-res}}(x_0, y_0, p(x, y), n) = p(x_0 \text{ div } 2^n, y_0 \text{ div } 2^n), \quad (42)$$

where  $n$  = resolution enlargement factor ( $n > 1$ ). The new resolution format is

$$M' = 2^n \cdot M \text{ and } N' = 2^n \cdot N, \quad (43)$$

where  $M \times N$  = resolution format of  $p(x, y)$ ,

$M' \times N'$  = resolution format of  $p'(x, y)$ .

Increasing resolution does not change appearance of an image but may be necessary for matching resolution formats of different imaging software or input/output devices.



Figure 2-17. 128x120 image of figure 1-3.



Figure 2-18. 64x60 image of figure 1-3.

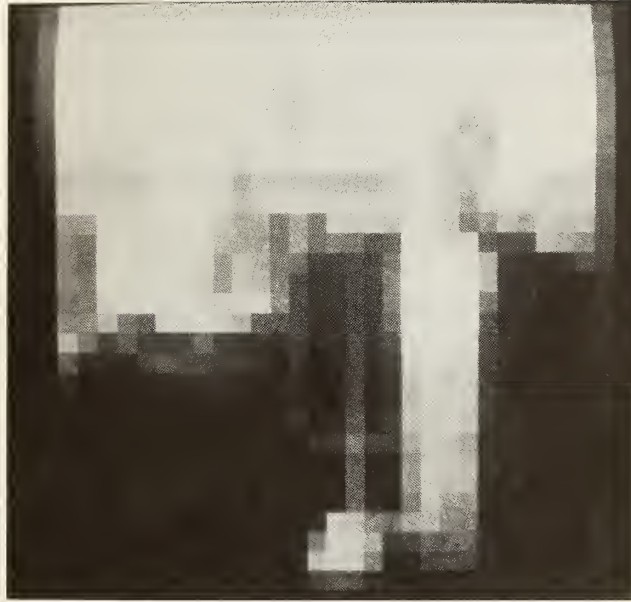


Figure 2-19. 32x30 image of figure 1-3.



Figure 2-20. 16x15 image of figure 1-3.

## 2.3 Two-Picture Functions

Two-picture functions are image processing functions that compute each pixel as a function of the corresponding pixels of two pictures,  $p_1(x,y)$  and  $p_2(x,y)$ . The functions have the form,

$$p_1(x_0,y_0) = f_{\text{two-picture}}(p_1(x_0,y_0), p_2(x_0,y_0)), \quad (44)$$

where  $p_1(x_0,y_0)$  is the current pixel being processed in the picture  $p_1(x,y)$ . (In this algorithm,  $p_1(x,y)$  is replaced by the new array.) Examples of two-picture functions include averaging, addition, subtraction, multiplication, and logical functions. The averaging function is the only two-picture function included in the software and is a very useful method of noise reduction.

These functions scan the entire picture and process each pixel individually. Both pictures must have the same resolution format. In general these functions are quick and usually do not require complex computation.

### 2.3.1 Averaging

Averaging is creating a picture that is the pixel-by-pixel average of two pictures. The general form of the equation is

$$f_{\text{average}}(p_1(x_0,y_0), p_2(x_0,y_0)) = \frac{p_1(x_0,y_0) + p_2(x_0,y_0)}{2}, \quad (45)$$

where  $p_1(x,y)$  = current picture in memory,  
 $p_2(x,y)$  = input picture file.

The purpose of this process is to reduce random noise such as video camera noise. Averaging two or more pictures can reduce this type of noise significantly.

Using only a two-picture function, we may also average any number of pictures that is a power of 2. For example, four pictures can be averaged in three steps as follows,

$$\begin{aligned}
p_3(x_0, y_0) &= f_{\text{average}}(p_3(x_0, y_0), p_4(x_0, y_0)), \\
p_1(x_0, y_0) &= f_{\text{average}}(p_1(x_0, y_0), p_2(x_0, y_0)), \\
p_1(x_0, y_0) &= f_{\text{average}}(p_1(x_0, y_0), p_3(x_0, y_0)), \quad (46)
\end{aligned}$$

where  $p_1(x, y)$  becomes the average of the four pictures  $p_1(x, y)$ ,  $p_2(x, y)$ ,  $p_3(x, y)$ , and  $p_4(x, y)$ .

Figure 1-3 shows the result of averaging four images similar to figure 2-21. The eye is a marvelous averaging device; the image looks fine viewed live on a video screen but is noisy when digitized and frozen, as in figure 2-21. Figure 2-22 shows a noisy image of the output of a single-mode optical fiber. The image is obscured by randomly generated noise as a result of the low-light conditions. Figure 2-23 is the average of eight images taken separately but all similar to figure 2-22. The averaged image contains much less random noise.



Figure 2-21. Unaveraged image of figure 1-3.

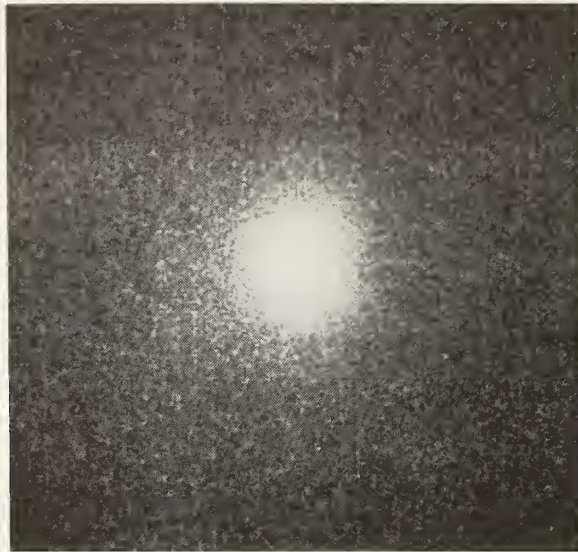


Figure 2-22. Noisy image of a Gaussian beam.

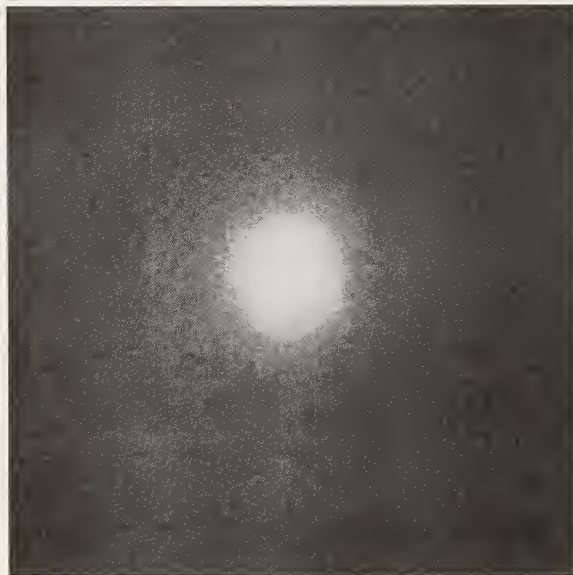


Figure 2-23. Averaged image of figure 2-22.

## 2.4 Image Information Display Tools

Image Information Display Tools compute and display information about a digitized image. Some of these tools are custom in that they are designed to meet the needs of a specific user. The Gaussian and g-profile curve-fitting tools are specially designed to meet the needs of characterizing beam spot radius of single-mode and multimode optical fibers. The histogram-of-intensity graph, line intensity graph, and three-dimensional intensity plot have much more universal applications.

### 2.4.1 Gaussian Curve Fitting

Two-dimensional Gaussian intensity distributions are found in lasers, optical fibers, and integrated optical devices. The purpose of the Gaussian curve fitting tool is to take a one-dimensional line of data from a digitized image and fit a Gaussian curve to it.

Since the dark pixel value of most video cameras is not 0, we usually want to bias the image with the negative of the dark level, that is, to clip the dc level. This dc level may be taken to be the peak value of the intensity histogram. Also, since these images are usually taken under low-light conditions we fit to the Gaussian function using only pixel values greater than a noise cutoff value,  $n$ , which is taken to be the maximum absolute value of the video and quantization noise. We subjectively set the value of  $n$  by looking at a line intensity scan of the beam image. After removal of the dark level, pixel values less than or equal to  $n$  are ignored in the curve fitting routine.

After biasing and deciding on an appropriate  $n$ , we calculate the centroid or first moment of the image. The centroid is the center position of a digitized picture whose coordinates are determined by the weighted average along each axis. The general form of the coordinates of the centroid [3] is

$$c_x = \left( \sum_{y=0}^N \sum_{x=0}^M x \cdot p'(x,y) \right) \cdot \left( \sum_{y=0}^N \sum_{x=0}^M p'(x,y) \right)^{-1},$$
$$c_y = \left( \sum_{y=0}^N \sum_{x=0}^M y \cdot p'(x,y) \right) \cdot \left( \sum_{y=0}^N \sum_{x=0}^M p'(x,y) \right)^{-1}, \quad (47)$$

where  $(c_x, c_y)$  = centroid of image,

$$p'(x,y) = \begin{cases} p(x,y) & p(x,y) > \eta \\ 0 & \text{otherwise,} \end{cases}$$

$\eta$  = noise cutoff value.

The general form of the one-dimensional Gaussian curve is

$$p(u) = A \cdot e^{-\alpha(u-c)^2}, \quad (48)$$

$$\text{where } u, c = \begin{cases} x, c_x & \text{for horizontal line fit at } y = c_y \\ y, c_y & \text{for vertical line fit at } x = c_x. \end{cases}$$

To fit a Gaussian curve to the data, we linearize by taking logarithms and use a linear least squares fit. In linear form,

$$\ln(p(u)) = \ln(A) - \alpha(u-c)^2,$$

$$y' = \ln(A) - \alpha x', \quad (49)$$

where  $y' = \ln(p(u))$  and  $x' = (u-c)^2$ . We now have the basic equation of a line,

$$y' = mx' + b \quad (50)$$

where  $m = -\alpha$  and  $b = \ln(A)$ . We use the method of least squares to determine the line of best fit with the formulas [4],

$$m = \frac{k \sum x_i' y_i' - \sum x_i' \sum y_i'}{k \sum (x_i')^2 - (\sum x_i')^2},$$

$$b = \frac{\sum (x_i')^2 \sum y_i' - \sum x_i' \sum x_i' y_i'}{k \sum (x_i')^2 - (\sum x_i')^2}, \quad (51)$$

$$\left. \begin{aligned} \text{where } y_i' &= \ln(p(u)) \\ x_i' &= (u-c)^2 \end{aligned} \right\} \quad \text{but not used if } p(u) < \eta,$$

$$k = \text{number of data points for which } p(u) > n$$

$$= \begin{cases} < M & \text{for horizontal fit} \\ < N & \text{for vertical fit.} \end{cases}$$

After the fit is completed we have

$$A = e^b \quad \text{and} \quad \alpha = -m. \quad (52)$$

We use these parameters to calculate the beam spot radius of the fitted data. The beam spot radius,  $r$ , is defined as the  $1/e^2$  point relative to the maximum of the intensity distribution. Using the  $\alpha$  parameter, we see that

$$1/e^2 = e^{-\alpha r^2}$$

and

$$r = (-\ln(1/e^2)/\alpha)^{1/2} \quad (53)$$

Examples of the Gaussian curve fitting tool can be seen in section 3.2.

#### 2.4.2 Multimode Curve Fitting (g-Profile)

The purpose of the multimode curve fitting tool is to fit a power-law profile, known as a g-profile, to a graded index multimode fiber with uniform cladding. The index profile is given as [5]

$$n(r) = n_0[1 - 2\Delta(r/a)^g]^{1/2}, \quad (54)$$

where  $\Delta = (n_0^2 - n_2^2)/2n_0^2$ .

This equation is made up of four parameters,  $n_0$ ,  $\Delta$ , core radius  $a$ , and exponent  $g$ . The parameters  $a$  and  $g$  are of most importance.

Since fitting a set of data to four parameters is not trivial, a simplification is needed. Following Cherin [6], we use the approximation,

$$\Delta n(r) = \Delta n[1 - (r/a)^g], \quad (55)$$

where  $\Delta n = n_1 - n_2$  and  $\Delta n(r) = n(r) - n_2$ .

This new equation now contains only three parameters,  $\Delta n$ ,  $a$ , and  $g$ .

Since we are interested in looking at digitized images of the near field of multimode fibers, the data are in the form of an intensity distribution,

$$I(r) = I_0[1 - (r/a)^g]. \quad (56)$$

Taking logarithms of both sides, we see that

$$\ln[1 - I(r)/I_0] = g \cdot \ln[r] - g \cdot \ln[a], \quad (57)$$

which follows the basic equation of a line,

$$y = mx + b, \quad (58)$$

where  $y = \ln[1 - I(r)/I_0]$ ,

$$m = g,$$

$$x = \ln[r],$$

$$b = -g \cdot \ln[a].$$

Using a least-squares fit, eq (51), we can solve for  $m$  and  $b$  for a given  $I_0$ . The  $g$ -parameter we seek is the slope  $m$ , and the core radius is

$$a = e^{-b/m}. \quad (59)$$

To find the center of the two-dimensional profile, a calculation of the centroid is performed, eq (47), similar to the Gaussian curve fit in section 2.4.1. From the coordinates of this centroid we can determine the center line, in either horizontal or vertical direction, and the center of that line from which we compute the radius.

In the EIA (Electronic Industries Association) standard FOTP-58 [7], only the data points in the 10 percent and 80 percent range of the total profile are used. In other words, valid data points used in the curve fitting are only those which fall between  $0.1I_0$  and  $0.8I_0$ .

$I_0$  cannot be obtained by a least-squares fit with only two parameters. Consequently, we optimize the fit with respect to  $I_0$  by looking at the quality of the fit, which is taken as the error,

$$E = \sum (I(r) - I_0[1 - (r/a)^g])^2. \quad (60)$$

We initially set  $I_0$  to be equal to  $I_{\max}$ , the maximum pixel intensity in the digitized image. We then minimize the error function,  $E$ , by monitoring  $E$  as  $I_0$  is changed and recalculating the parameters  $a$  and  $g$ . Once  $E$  is minimized, a final set of parameters for  $I_0$ ,  $a$ , and  $g$  is reached.

#### 2.4.3 Intensity Histogram

An intensity histogram is a two-dimensional graph of relative frequency versus the pixel intensity values, 0 to 255. The relative frequency is defined as the number of occurrences of a pixel intensity on a normalized scale. This scale is defined to be 1.0 at the most numerous pixel(s) and scaled accordingly for the remaining pixel values.

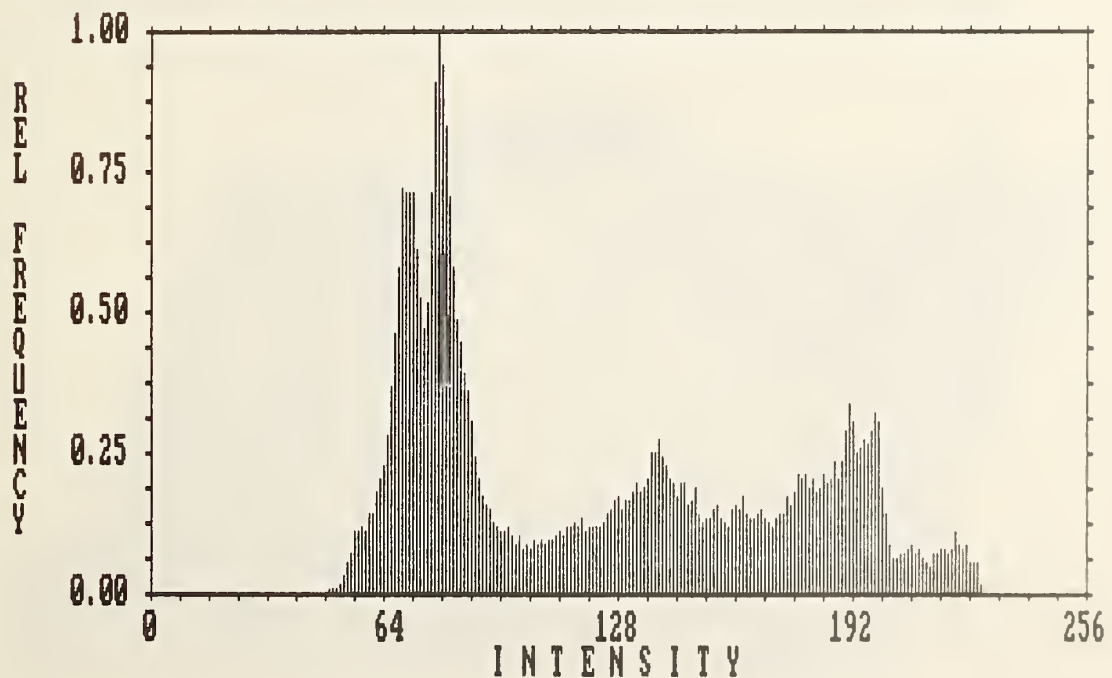


Figure 2-24. Intensity histogram of figure 1-3.

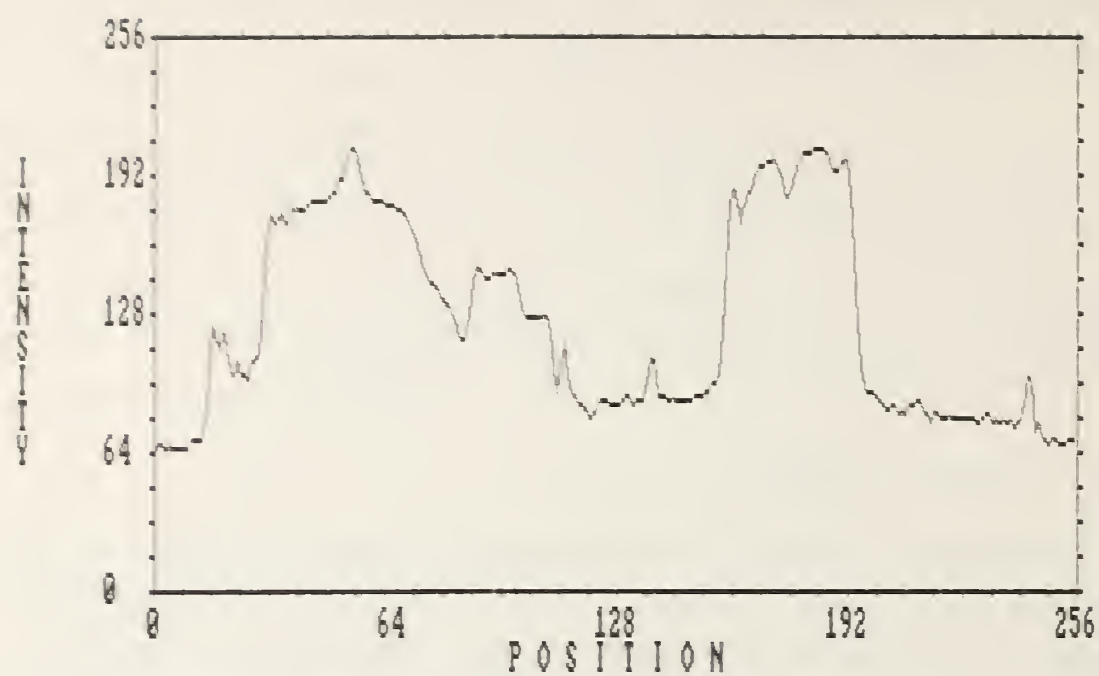


Figure 2-25. Line intensity scan of figure 1-3.

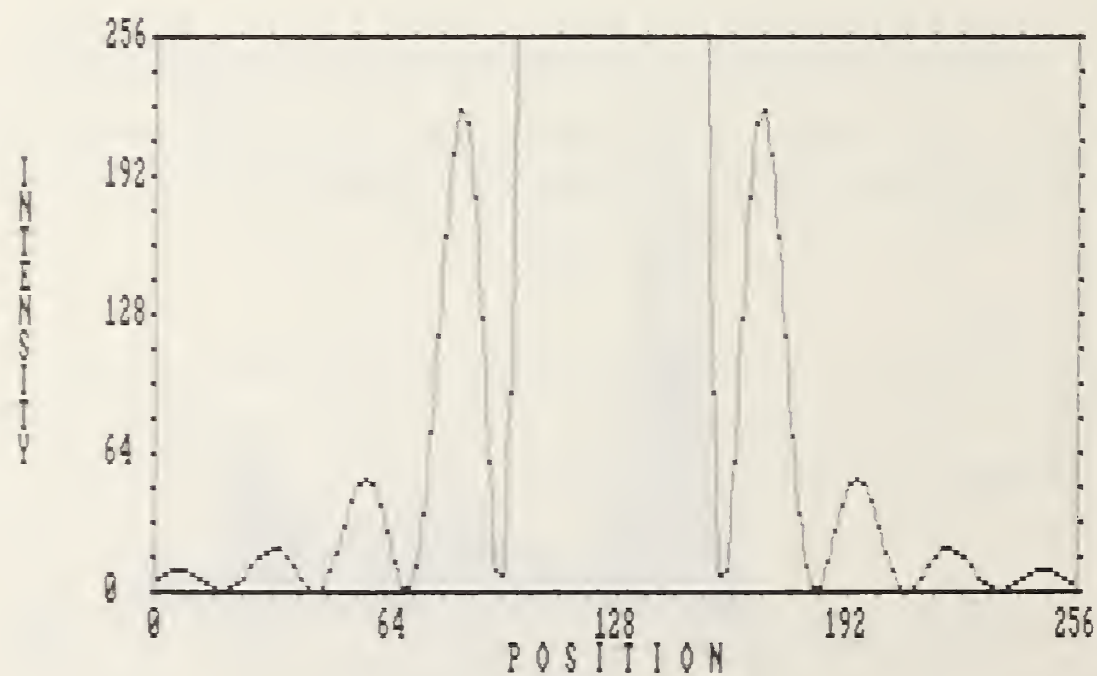


Figure 2-26. Line intensity scan of figure 2-11.

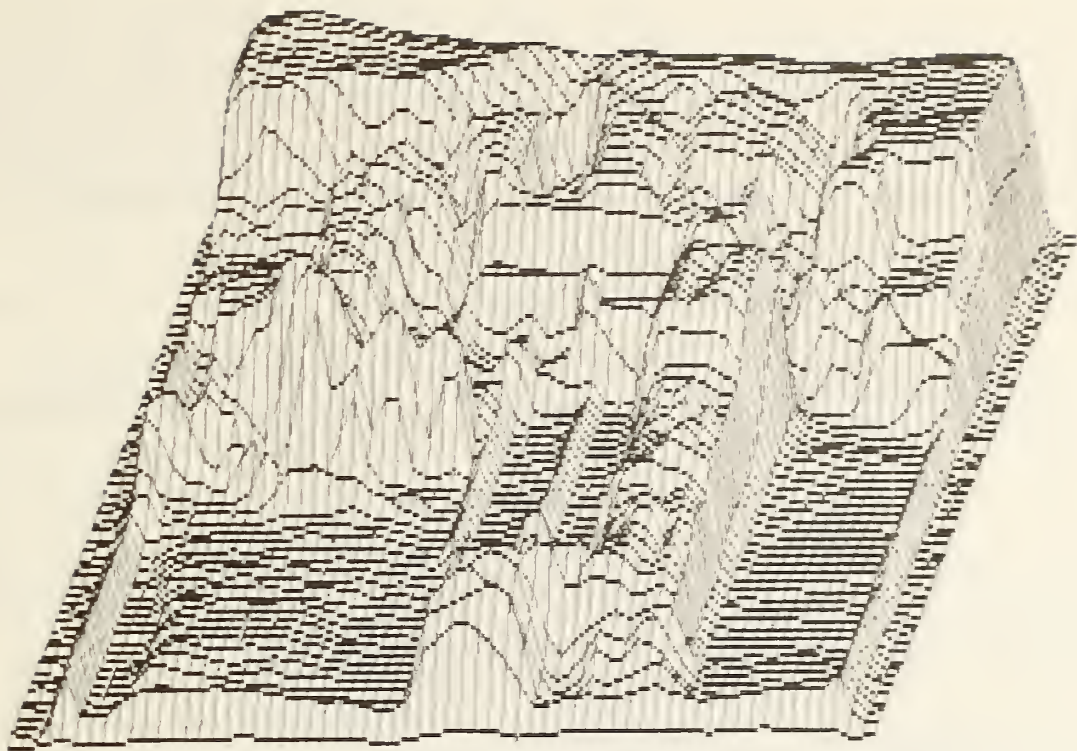


Figure 2-27. Three-dimensional intensity plot of figure 1-3.

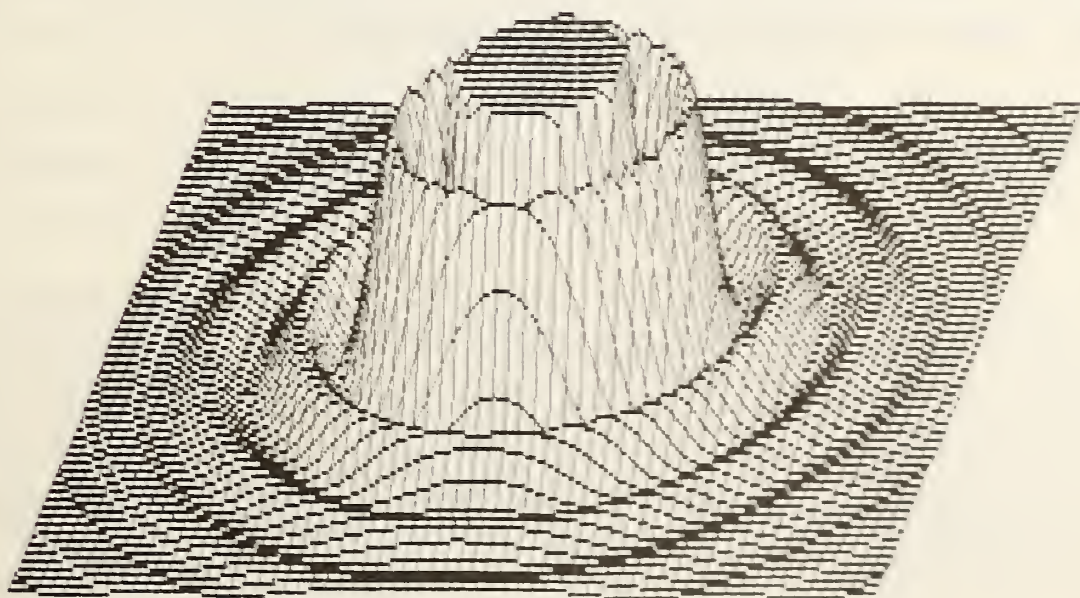


Figure 2-28. Three-dimensional intensity plot of figure 2-11.

Using a histogram we can look at the relative contrast range of an image. Also computed are the highest, lowest, average, and highest relative frequency (or peak) pixel. Figure 2-24 shows a histogram of figure 1-3. The highest pixel value is 227, the lowest is 49, the average is 121, and the peak is 79.

#### 2.4.4 Line Intensity Scan

The line intensity scan tool allows us to graph the pixel intensity values across a specified line in either the horizontal or vertical direction.

Figure 2-25 shows a horizontal intensity scan of line 120 of figure 1-3. Figure 2-26 shows a horizontal intensity scan across line 120 of figure 2-11. Using these scans we can analytically monitor the results of image processing functions and read image data with reasonable accuracy.

#### 2.4.5 Three-Dimensional Intensity Plot

The three-dimensional intensity plot generates a visual surface whose height is determined by the intensity levels of an image versus two-dimensional positions [8].

Figure 2-27 shows a three-dimensional plot of figure 1-3. Figure 2-28 shows a three-dimensional plot of figure 2-11. Using these plots we can better visualize the image intensity distributions.

### 3. Applications of the Software

#### 3.1 Creating a Model Image for a Hybrid Computer-Optical Processing Experiment

The purpose of our experiment in hybrid computer-optical processing was to do real-time pattern recognition at video rates. The computer was responsible for preprocessing a model image. This model was then permanently stored as a Fourier transform hologram. Optical processing was then used in real time to recognize a live image. For complex enough images, optical processing will be much faster than the computer.

In this experiment we created a model image for real-time pattern recognition [9]. A liquid-crystal television (LCTV) was used as a real-time transparency to display digitized images. We used an optical processor in series with a converging beam processor [1] to display the Fourier transform of a specially prepared model and generated its Fourier transform hologram. Actual images were then displayed on the LCTV, and their (optical) Fourier transforms illuminate the hologram. The convolution theorem, eq (9), shows that we can obtain the convolution of the model with the actual image as our method of image recognition. The purpose of this experiment was to use the computer's advantage in digital image input/output processing and also take advantage of optical processing's superior convolution speed.

As a test 256x240 model image, we used an ordinary pair of pliers as in figure 3-1. We chose the pliers for their distinct shape and applicability to image recognition in manufacturing. The shiny surface of the pliers required that we use very diffuse and uniform illumination with a black background to bring out the plier's overall shape rather than accent the reflected light from its contours.

Since the background of figure 3-1 is not perfectly black, the noise cleaning function of section 2.2 was used to remove noise and spurious bright spots. To remove the background entirely, we used the thresholding function and specified a threshold range to set that area to 0, being careful not to erase any detail of the pliers. The resulting image is shown in Figure 3-2.

To make the outline shape of the pliers as distinct as possible we used the thresholding function once again and set all the pixels of the pliers to 255. The resulting image is shown in figure 3-3.

From the model image we wished to obtain the edge pattern of figure 3-3. To do this we normally convolve the object with a high-pass convolution kernel, given in eq (12). In this case the image will be projected on an LCTV with resolution of only 140 x 120 pixels. Since this lower resolution format does not match exactly any of the formats handled by this software, the image resolution of the model image remained at 256 x 240 to achieve maximum detail. To compensate for the larger pixels of the LCTV we used a larger convolution kernel,



Figure 3-1. 256x240 digitized image of pliers.



Figure 3-2. Background processed image of figure 3-1.

$$k_{\text{model}} = \begin{vmatrix} 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{vmatrix}. \quad (61)$$

This convolution kernel is analogous to eq (12), but produces an outline of the image that is three times wider, which is large enough for the LCTV. The final model image is shown in figure 3-4. Details of the pattern recognition experiment are given in reference [9] and are reproduced as appendix C.

### 3.2 Analyzing Near- and Far-Field Images for an Integrated Optics Experiment

The propagation of light in the form of a Gaussian beam has great interest in the fields of optical fibers and integrated optics. The important parameters of a Gaussian beam are the radius of curvature,  $R(z)$ , of the spherical wave and its spot size,  $w$ . With these parameters we can predict the propagation characteristics of the beam.

The spot size is defined as the radial distance from the center to the  $1/e$  point of the Gaussian beam's amplitude distribution. In a homogenous medium the form of the Gaussian beam is [10]

$$E(x,y) = E_0 \cdot \frac{w_0}{w(z)} \cdot \exp\{-i[kz - \eta(z)] - r^2(\frac{1}{w^2(z)} + \frac{ik}{2R(z)})\},$$

$$E(x,y) \propto E_0 \cdot \frac{w_0}{w(z)} \cdot \exp\{-\frac{r^2}{w^2(z)}\}, \quad (62)$$

where  $r = (x^2 + y^2)^{1/2}$  = radial distance,

$R(z)$  = radius of curvature of the spherical wave,

$w(z)$  = spot size at distance  $z$ ,

$w_0$  = minimum spot size (at  $z = 0$ ),

$E_0$  = amplitude coefficient,

$k = \frac{2\pi n}{\lambda}$  = propagation constant of the medium,

$n$  = refractive index of the medium,

$\lambda$  = optical wavelength.



Figure 3-3. Foreground processed image of figure 3-2.

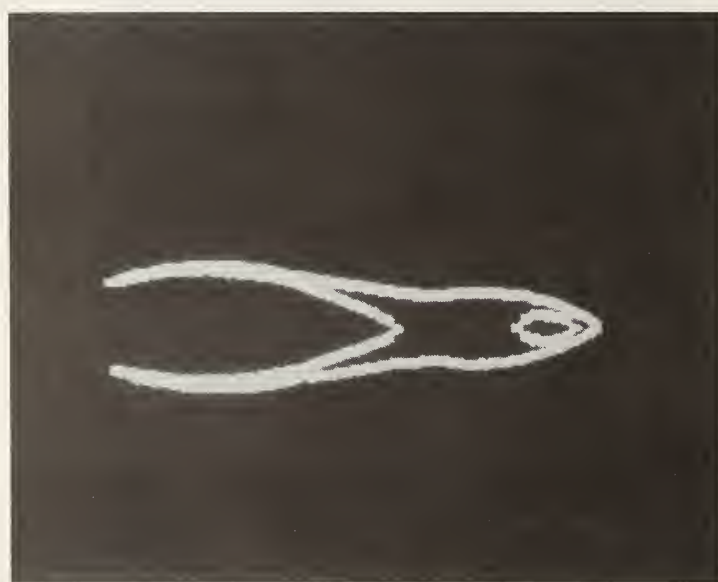


Figure 3-4. Convolution model of figure 3-1.

Since we are interested in intensity distributions as seen by the video camera, the beam spot size,  $w$ , is the  $1/e^2$  point of the intensity distribution. The general form of the intensity distribution [10] is

$$I(x,y) \propto E^2(x,y) \propto E_0^2 \cdot \frac{w_0^2}{w^2(z)} \cdot \exp\left(-\frac{2r^2}{w^2(z)}\right),$$

or

$$I(x,y) \propto A \cdot \exp(-\alpha r^2) \quad (63)$$

where  $A = E_0^2 \cdot \frac{w_0^2}{w^2(z)}$  and  $\alpha = 2/w^2(z)$ .

The  $A$  and  $\alpha$  terms correspond to the parameters discussed for the Gaussian curve-fitting tool in section 2.4.

Using the Gaussian curve-fit tool we can measure the spot size,  $w$ , at a certain distance. Subsequent spot sizes can be then be predicted by the equation [10],

$$w^2(z) = w_0^2 \cdot \left(1 + \left(\frac{\lambda z}{\pi w_0^2 n}\right)^2\right), \quad (64)$$

or

$$w(z) \approx \frac{\lambda z}{\pi w_0 n}$$

when  $\left(\frac{\lambda z}{\pi w_0^2 n}\right)^2 \gg 1$ .

In the first experiment, as shown in figure 3-5, we looked at a near-field image of a beam that is approximately Gaussian. We used a single-mode optical fiber that nominally has a 5  $\mu\text{m}$  core diameter and a numerical aperture of 0.11. The fiber is illuminated by a quartz halogen lamp with a 851 nm narrowband filter. The other end of the fiber is then magnified by successive 40X and 10X microscope objectives. The near-field image was captured with a vidicon equipped with automatic gain control.

The digitized image of the near-field spot is shown in figure 3-6 with a resolution of 256 x 240. Averaging four similar images is necessary to reduce the video noise resulting from the low-light conditions. Figure 3-7 is the

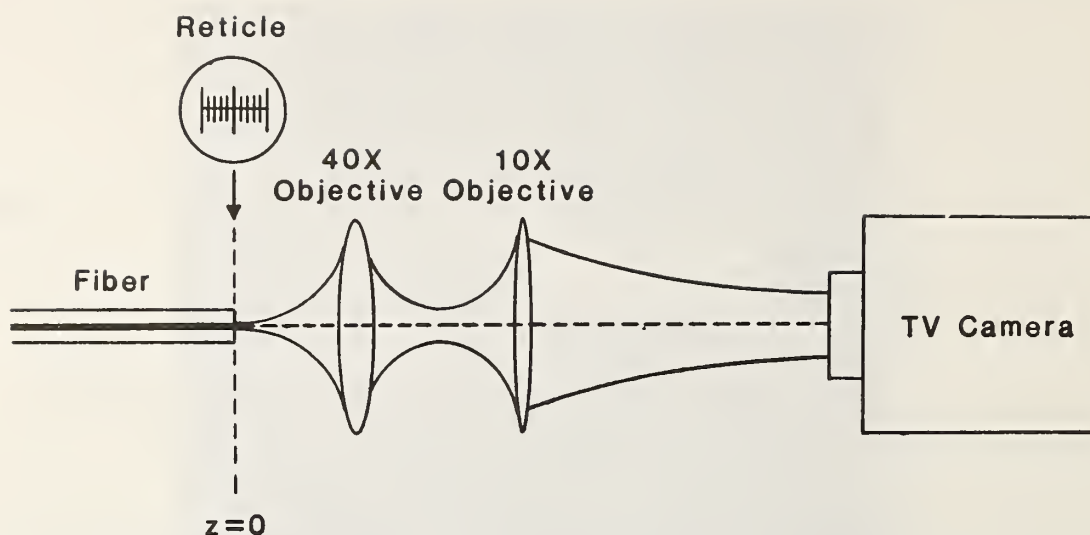


Figure 3-5. Experimental setup for the near-field image of an optical fiber.

image of a precision stage micrometer placed in the plane  $z = 0$ . The lines are  $2\text{ }\mu\text{m}$  apart, so the screen has a total horizontal length of  $24\text{ }\mu\text{m}$ .

To obtain the beam spot radius we first reduced the resolution of the near-field image to  $64 \times 60$  using the resolution changing function. This allowed for adjacent lines to be averaged to reduce possible errors. Figure 3-8 is the intensity histogram of the near-field image and shows good contrast. We took the peak histogram intensity value, 83, as the average value of the dark background with 0 intensity. The intensity of the image of the near field was then reduced by -83 using the biasing function.

Figure 3-9 is the Gaussian curve fit to the near-field beam. The continuous line represents the fitted curve and the dots represent the actual data points. A noise cutoff value was  $\eta = 20$  (sect. 2.4). The value of  $\eta$  is a subjectively chosen value of the maximum noise shown in figure 3-9. The

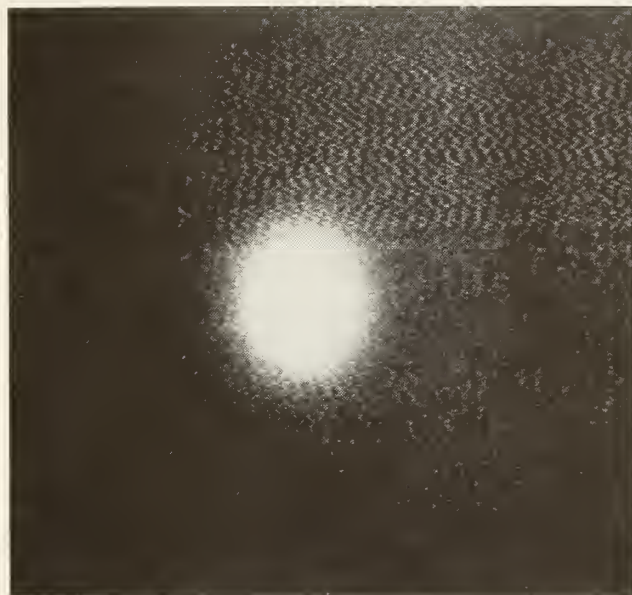


Figure 3-6. Near-field image of an optical fiber.



Figure 3-7. Image of scaling reticle for near field.

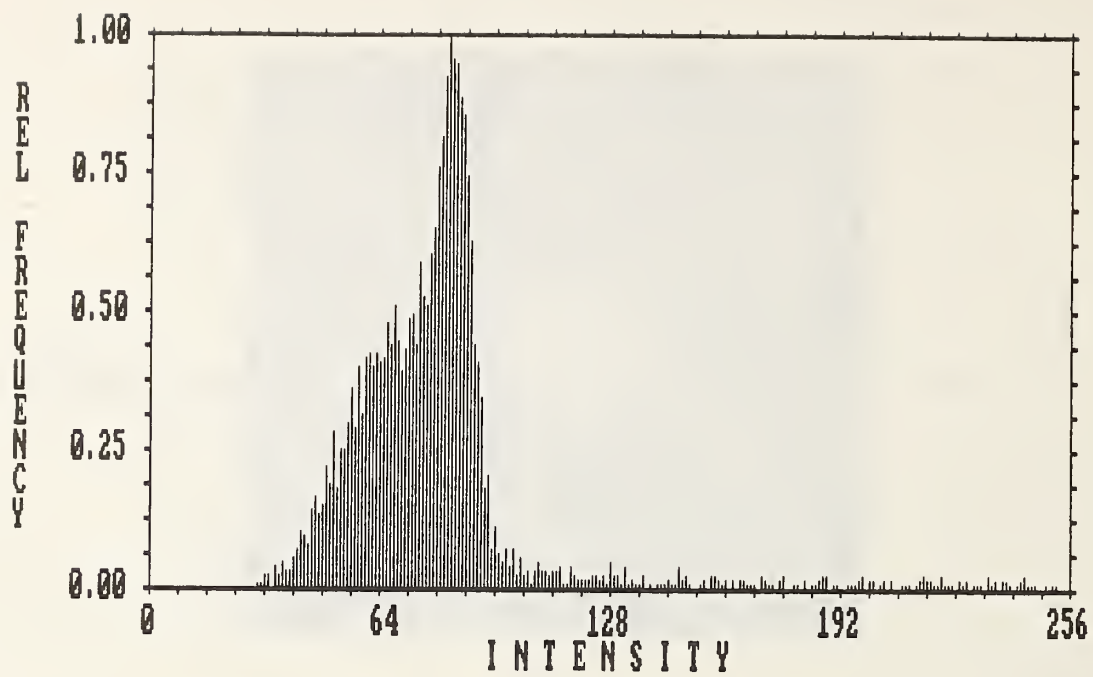


Figure 3-8. Intensity histogram of near-field image.

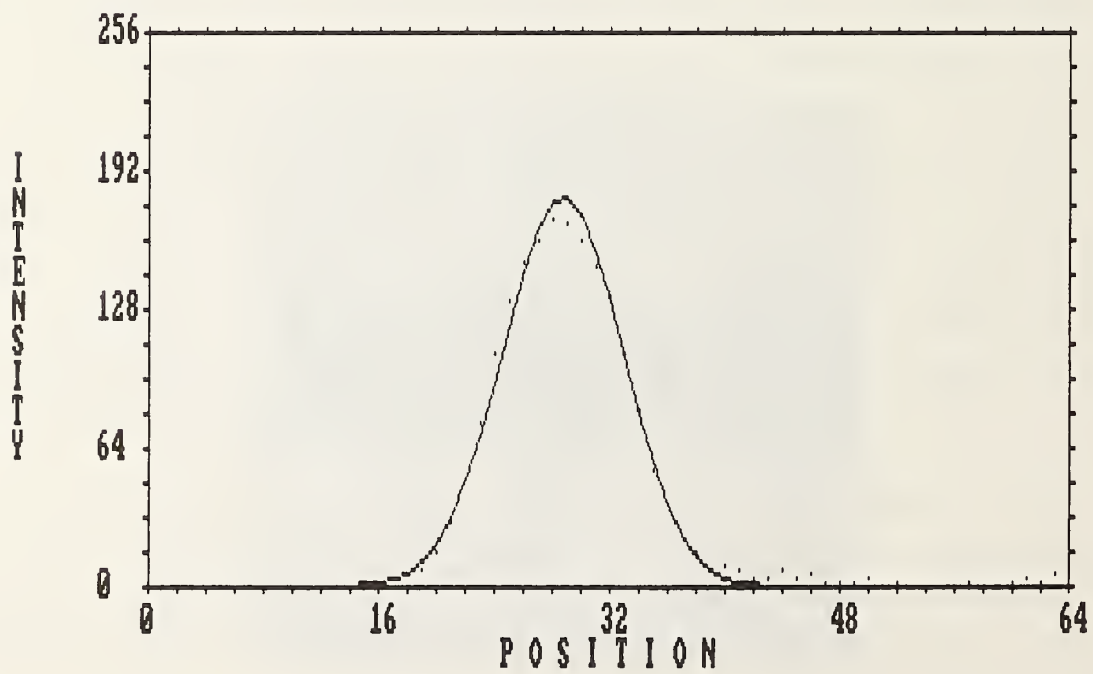


Figure 3-9. Gaussian curve fit of near-field image.

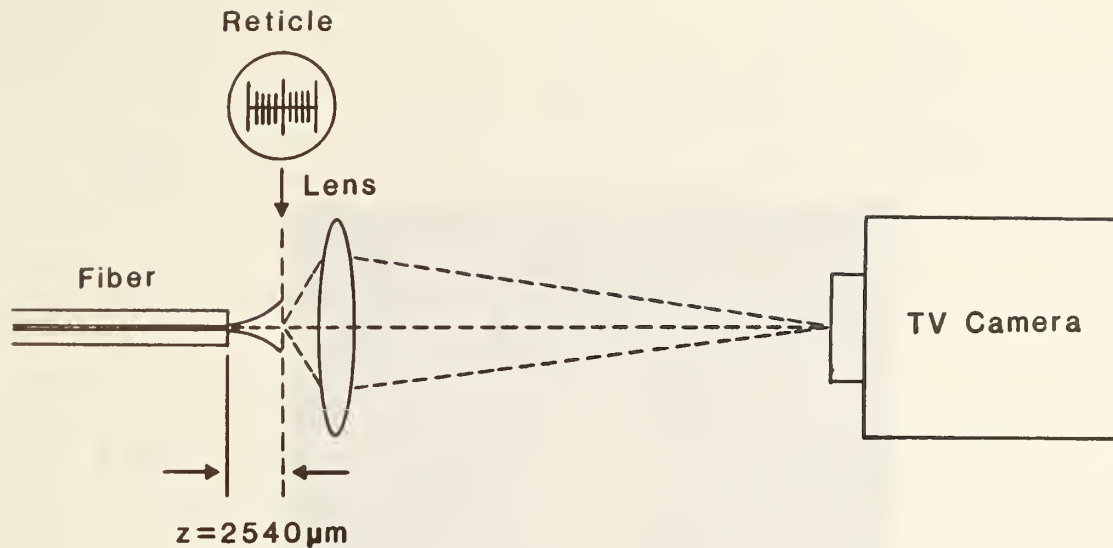


Figure 3-10. Experimental setup for the far-field image of an optical fiber.

computed beam spot radius is 8.2891 pixels with 64 pixels across the horizontal screen length. The spot radius at  $z = 0$  is

$$w_0 = \left( \frac{8.2891 \text{ pixels}}{64 \text{ pixels}} \right) \cdot (24 \mu\text{m}) = 3.108 \mu\text{m}.$$

In the second experiment, shown in figure 3-10, we looked at the far-field image. We used the same single-mode optical fiber and 851 nm illumination. The far-field beam of the fiber at a distance  $z = 2540 \mu\text{m}$  was magnified by a single lens and imaged onto the vidicon with automatic gain control.

The digitized image of the far-field beam spot is shown in figure 3-11 with a resolution of  $256 \times 240$ . Averaging eight similar images was necessary to reduce the video noise resulting from the low-light conditions. Figure 3-12 is the image of a stage micrometer placed in the plane  $z = 2540 \mu\text{m}$ . Lines are  $10 \mu\text{m}$  apart, so the screen has a total horizontal length of about  $950 \mu\text{m}$ .

To obtain the beam spot radius we first reduced the resolution of the near-field image to  $64 \times 60$  using the resolution changing function. This allows adjacent lines to be averaged to reduce possible errors. Figure 3-13 is the intensity histogram of the near-field image, and shows moderate contrast. Again, we take the peak histogram intensity value, 65, as the average value of the dark background with 0 intensity. The intensity of the image of the near field was therefore reduced by -65 with the biasing function.

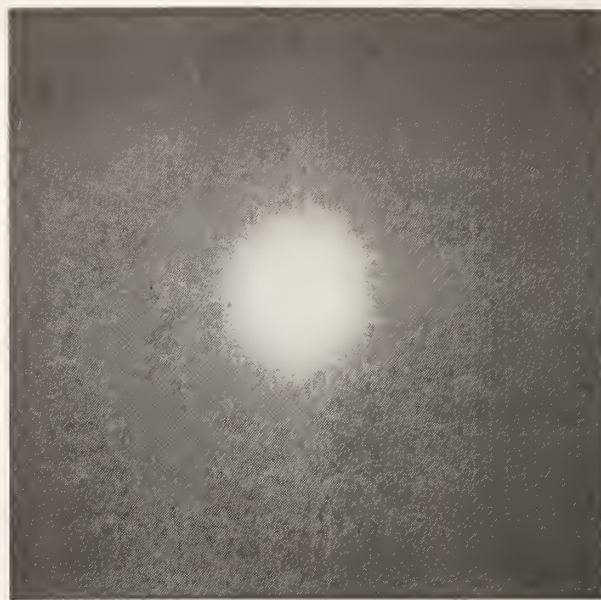


Figure 3-11. Far-field image of an optical fiber.

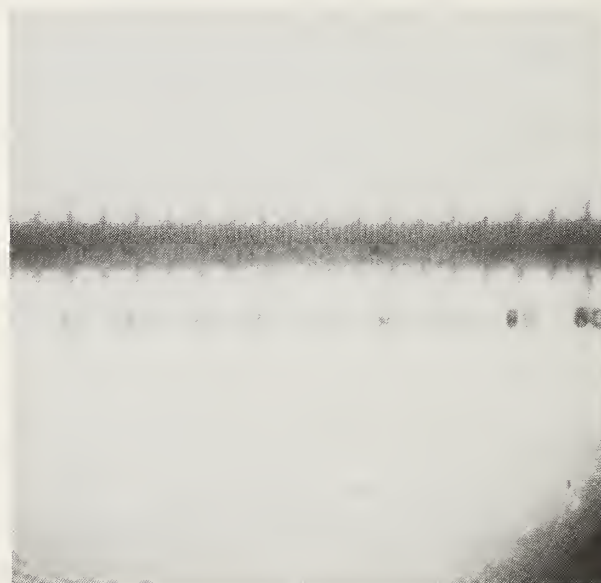


Figure 3-12. Image of scaling reticle for far field.

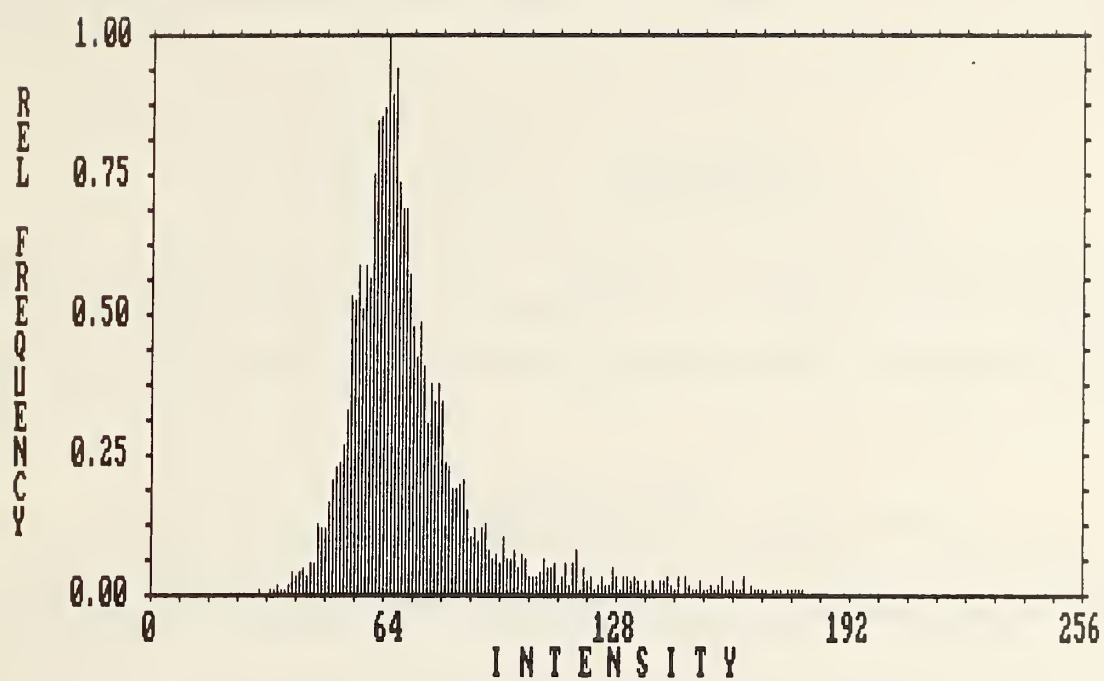


Figure 3-13. Intensity histogram of far-field image.

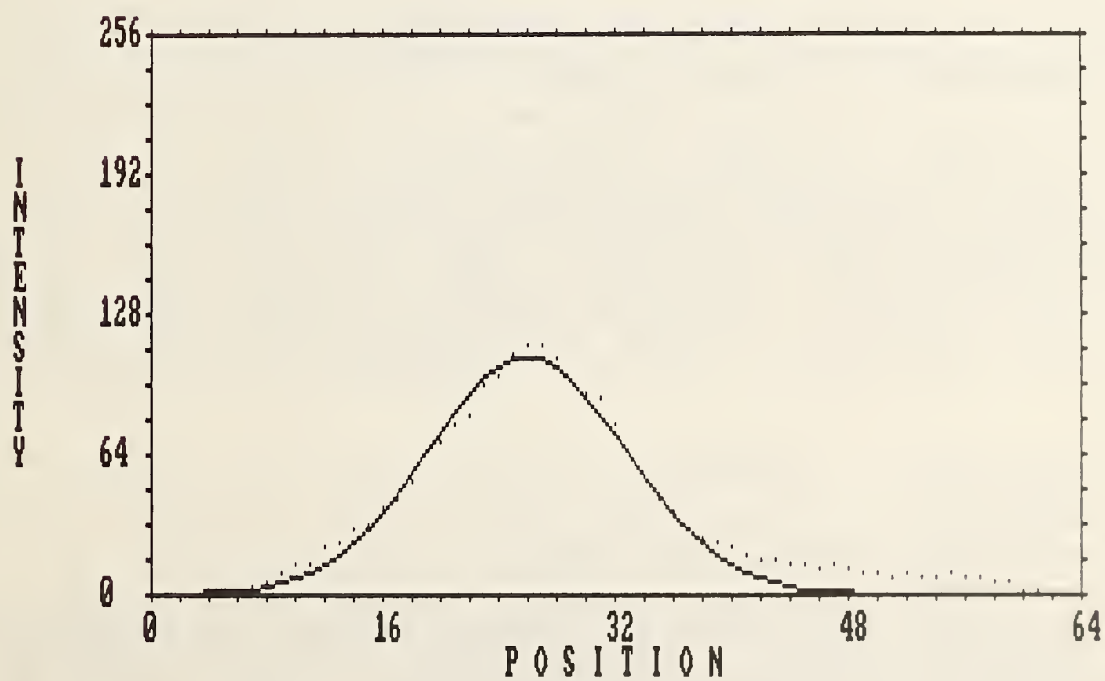


Figure 3-14. Gaussian curve fit of far-field image.

Figure 3-14 is the Gaussian curve fit to the far-field beam with a noise cutoff value of 32. The beam spot radius is 13.629 pixels with 64 pixels across the horizontal screen length. The spot radius at  $z = 2540 \mu\text{m}$  is therefore

$$w_0 = \left( \frac{13.629 \text{ pixels}}{64 \text{ pixels}} \right) \cdot (950 \mu\text{m}) = 202.3 \mu\text{m}.$$

We can test the results of the near- and far-field experiments by predicting the far-field spot radius from the spot radius in the near field. Using the formula for the propagation of the Gaussian beam, eq (64), we see that

$$w(z) \approx \frac{\lambda z}{\pi w_0 n} = \frac{(0.851 \mu\text{m})(2540 \mu\text{m})}{(3.1415)(3.108 \mu\text{m})(1.0)} = 221.4 \mu\text{m},$$

whereas we have measured a value of  $202.3 \mu\text{m}$ . The relative error is thus

$$\% \text{ error} = \frac{(221.4 \mu\text{m}) - (202.3 \mu\text{m})}{202.3 \mu\text{m}} \cdot 100\% \sim 9\%.$$

The near- and far-field images produce consistent results.

The error incurred in this set of preliminary experiments is a result of many contributions. The small distance  $z$  was hard to measure with great accuracy but can be increased through the use of longer focal length lenses. The rectilinearity and, especially, the linearity of responsivity of the vidicon camera are questionable. These could be improved greatly with a CCD (charge coupled device) array camera but light from the beam signal was too low for our CCD camera. This may be improved with a broader band interference filter or a laser source. Distortion in the lenses is also subject to scrutiny.

## 4. Conclusion

### 4.1 Conclusions and Future Applications

The purpose of this thesis was to develop and test image processing software designed for optical engineering applications. Image processing is a relatively new and rapidly advancing field. Present applications of image processing include aerial photography analysis, medical analysis, photographic

processing, robotic image recognition, and inspection, to name a few. Most of the software and systems available today are designed with universal applications in mind to meet the widest sector of potential customers in this new area. The software in this thesis is partly universal in nature, but has been written with optical engineering applications in mind.

Image processing has many strengths which are advantageous to optical engineering. By using software and mathematically modeling optical devices we can simulate optical processing without the actual devices. This is important in the design and simulation of electrical and electronic components in very large scale integrated (VLSI) electronic chips. Simulating optical processing may prove to be even more important.

Since image processing deals with digital images we can process such images with great numerical accuracy. Real optical components are also image processors but are quite costly to make with high quality and are time consuming to implement. Image processing offers a cheaper and faster alternative with greater possibilities. The image processing functions described in section 2 process images without coherent optics. With more sophisticated hardware and algorithms the software may actually do a better job than real optical devices. Computer processing can also perform functions that cannot be performed optically, and sometimes these can be used to advantage in coherent optical processing.

As shown in section 3.2, we can also use image processing in fiber and integrated optics; other optical devices can also be analyzed with computer processing. This is important in both the characterization and testing of experimental and newly manufactured optical devices. The output of whole optical systems can also be analyzed.

The software in this thesis serves as an introduction to image processing for optical engineering. Possible future directions for a project of this type involve a system with greater resolution and computing capacity. Today's digital design is heavily involved in working to create faster and more specialized hardware to meet these goals. With greater computing power more complex and challenging problems can be investigated. There is also the need for better input and output devices for image processing systems. High resolution

digitizing cameras and scanning devices are very expensive and limited. Similarly, high resolution displays and printers are advancing but still have a long way to go. In particular, a cheap, fast, high resolution coherent display would be most useful. Liquid crystal televisions are cheap and available, but have low resolution, contrast, and speed. Ferroelectric crystal displays are more promising in these areas but are expensive and not fully developed at present. For both input and output devices there is a need for rectilinearity and linearity of responsivity to ensure high image fidelity and measurement accuracy.

## 5. References

- [1] J. W. Goodman, Introduction to Fourier Optics, (McGraw-Hill, San Francisco, 1968), Chaps. 2 and 7. M. Young, Optics and Lasers, 3rd ed. (Springer, New York, 1986), Chap. 6. W. T. Cathey, Optical Information Processing and Holography, (Wiley-Interscience, New York, 1974), Chaps. 2 and 7.
- [2] W. K. Pratt, Digital Image Processing, (Wiley-Interscience, New York, 1978), p. 319.
- [3] F. P. Beer, E. R. Johnson, Jr., Mechanics for Engineers, 3rd ed. (McGraw-Hill, New York, 1976), pp. 173-174.
- [4] C. E. Swartz, Used Math, (Prentice-Hall, New Jersey, 1973), p. 28.
- [5] M. Young, Optics and Lasers, 3rd ed. (Springer, New York, 1986), Chap. 9.
- [6] A. H. Cherin, P. J. Rich, and S. C. Mettler, "Measurement of the Core Diameter of Multimode Graded-Index Fibers: A Comparison of Transmitted Near-Field and Index Profiling Techniques," IEEE J. Lightwave Technol. LT-1 (2), 302-311 (June 1983).
- [7] Electronic Industries Association, FOTP-58, 2001 Eye St. N.W., Washington, DC 20006.
- [8] D. Hearn, M. P. Baker, Computer Graphics for the IBM Personal Computer, (Prentice-Hall, New Jersey, 1983), p. 248.

- [9] M. Young, M. B. Weppner, "Hybrid computer-optical processing with inexpensive liquid crystal television," Proc. SPIE 700, 146-153 (1986).
- [10] A. Yariv, Optical Electronics, 3rd ed. (Holt, Rinehart, and Winston, New York, 1985), p. 29.
- [11] The trade names are necessary to make the programs useful to the reader. Their use does not imply endorsement by the National Bureau of Standards.



## Appendix A. User's Manual

### 1. Introduction

#### 1.1 Software Description

IMPROC is an image processing software designed primarily for optical engineering applications. It contains a unique set of image processing tools not found in other current software.

IMPROC is generic in the sense that it works on ordinary MS-DOS data files. IMPROC is designed to be used in conjunction with a video image digitizer or frame-grabber. Picture files created by the frame-grabber are used as the input to IMPROC. IMPROC can manipulate and display information about these files using the various image processing tools. These altered files can be stored again as picture files, which can be subsequently displayed by the frame grabber.

### 2. Program Information

#### 2.1 Picture File Specifications

Picture files are the main means of storage of a digitized image. Each pixel of the digitized image is represented by one byte. This allows for the capability of up to 256 levels of intensity (gray scale) per pixel. The picture file consists of a linear array of pixels which represent the two-dimensional image. The file has a resolution format which describes the vertical and horizontal resolution of the image. The valid resolution formats allowed by IMPROC are

Resolution format (vert. by horiz.)	File size (bytes)
16x15	240
32x30	960
64x60	3,840
128x120	15,360 (15K)
256x240	61,440 (60K)

IMPROC allows the user to choose between detail and computational speed by changing resolution formats. Input picture files with resolution formats not listed above are illegal. Illegal files cannot be used and will prompt an error when attempted to be read by IMPROC.

Also throughout the software certain conventions about directional indices and implied center points are assumed.

Res. Format	Horiz. Indices	Vert. Indices	Center
16x15	0,1,...,15	0,1,...,14	(8,7)
32x30	0,1,...,31	0,1,...,29	(15,14)
64x60	0,1,...,63	0,1,...,59	(31,29)
128x120	0,1,...,127	0,1,...,119	(63,59)
256x240	0,1,...,255	0,1,...,239	(127,119)

These conventions are necessary for correct interpretation and manipulation in center dependent functions such as Fourier transform and magnification.

## 2.2 Program Execution

The IMPROC program is initiated by simply typing IMPROC. This calls the MS-DOS batch file IMPROC.BAT, which is the heart of the IMPROC software. This file controls the program flow of the three IMPROC execution files, IMPROC#1.EXE, IMPROC#2.BAS, and IMPROC#3.EXE.

IMPROC#1.EXE is an executable compiled C program which contains all the picture file manipulation code and is by far the largest of the three. This portion of the code is written in C to take advantage of its speed and program flow, which are necessary for the long and complex image processing algorithms.

IMPROC#2.BAS is an interpreted BASIC program used for two dimensional graphing on the graphics monitor. IMPROC#3.EXE is an executable compiled BASIC program used for three-dimensional plotting on the graphics monitor. These two programs were written in BASIC because of the lack of convenient

graphic capabilities in the C language. The second BASIC file, IMPROC#3.EXE, is compiled to speed up the tedious process of plotting.

Information is passed among the three programs by temporary data files, IMPROC#1.PIC, IMPROC#2.DAT, and IMPROC#3.DAT. IMPROC#1.PIC is the current picture being used by IMPROC#1.EXE and is stored when a call to IMPROC#2.BAS or IMPROC#3.EXE is made. IMPROC#2.DAT is the graphing data created by IMPROC#1.EXE and is used by the graphing program IMPROC#2.BAS. IMPROC#3.DAT is the plotting data created by IMPROC#1.EXE and is used by the plotting program IMPROC#3.EXE.

## 2.3 Program Compilation

To make changes, additions, or corrections in the software, we need to know how to compile the program source code. The program is written C using Lattice C Rev. 2.14 [11]. The program should be compiled under the "d" model which stresses a moderately sized program which works with a large amount of memory. The batch file, CCFFT.BAT, controls the compilation process

```
lcl %1 -mD
lc2 %1
link cd+fft87+%1,%1,nul,lcmd+lcd
```

To compile we simply need to type CCFFT IMPROC#1 and return. The machine code Fourier transform program, FFT87.C (Rapid Imaging Software [11]), is automatically linked. Compilation takes the source file, IMPROC#1.C, and generates the executable file IMPROC#1.EXE.

## 3. Commands

### 3.1 Image Commands

#### 3.1.1 Average

COMMAND: i [RETURN] a [RETURN]

FUNCTION:  $p(x) = (p(x) + \text{input}(x)) / 2$

DESCRIPTION: Averages the individual pixels of the current picture in memory with an input picture file. The result becomes the current picture.

### 3.1.2 Bias

COMMAND: i [Return] b [Return]

FUNCTION:  $p(x) = p(x) + \text{bias}$

DESCRIPTION: Adds an input integer value to each pixel. Valid range is -255 to 255. Biased values >255 are set to 255 and values <0 are set to 0.

OPTION(S): The auto-bias option computes a histogram and automatically biases (negatively) with the peak relative frequency pixel value.

### 3.1.3 Convolution

COMMAND: i [RETURN] c [RETURN]

FUNCTION:  $p(x) = \text{cross correlation with input matrix}$

DESCRIPTION: Computes the cross correlation of the neighbors of each pixel with an input matrix and sets the value to that pixel.

The input matrix has two dimensions, each of which must have an odd integer length. The matrix is centered on the current pixel and the elements of the matrix must be integers. A matrix normalization value is automatically calculated to set the area of the matrix elements to 1. If the matrix has zero area the normalization constant is set to 1.

During convolution the elements of the matrix are multiplied with their respective pixel neighbors. The sum of these products is then normalized and biased (optional). Pixels off the edge will cause erroneous convolutions and should be ignored. Pixel values created which are >255 are set to 255 and values <0 are set to 0.

OPTION(S): The normalization value may be set manually but not to zero. Also a bias value may be added after the convolution of each pixel.

#### 3.1.4 Demagnify

COMMAND: i [RETURN] d [RETURN]

FUNCTION: Demagnify picture by 2X

DESCRIPTION: Shrinks the current picture by a factor of 2 in each direction and places it in the center. This process results in a 4X loss of information as averaging of four pixels is used to create each demagnified pixel. The pixels of the blank area around the demagnified picture are set to 0.

#### 3.1.5 Enhance Contrast

COMMAND: i [RETURN] e [RETURN]

FUNCTION:  $p(x) = (255/(\text{highest}-\text{lowest})) * (p(x)-\text{lowest})$

DESCRIPTION: Increases the contrast over a specified range [lowest, highest] to full contrast [0,255]. Pixels values >255 are set to 255 and values <0 are set to 0.

OPTION(S): The autoscale option automatically finds the lowest and highest values to enhance the contrast to full scale without clipping.

#### 3.1.6 Fourier Transform

COMMAND: i [RETURN] f [RETURN]

FUNCTION: creates a 2-D FFT of picture.

DESCRIPTION: Creates a two-dimensional Fourier transform using a one-dimensional fast Fourier transform algorithm (FFT). The transform assumes that the picture is an intensity distribution made up of real components. The square root of each pixel is taken to represent the light amplitude. First the function processes the picture in the vertical direction to obtain the vertical transforms. Second the function processes in the horizontal direction to obtain the horizontal transforms. The result is taken to be the square of the real plus the square of the imaginary components which represent the Fourier transform intensity distribution.

OPTION(S): The user can make the choice of obtaining either the forward or inverse Fourier transform.

The user can also select an appropriate power-of-2 Fourier plane magnification to get a reasonably sized output transform relative to the input

picture. Fourier plane magnification results in the use of larger FFT arrays and subsequently requires more computation.

Finally the Fourier transform must be scaled to fit with the pixel limits [0,255]. The maximum point of the transform is taken to be 255. Using a scaling factor provides that the output can be scaled to observe finer detail not found in the dynamic range of the pixels. The height of the intensity pattern can then be taken to be the maximum point divided by the scale factor.

### 3.1.7 Magnify

COMMAND: i [RETURN] m [RETURN]

FUNCTION: Magnify picture by 2X.

DESCRIPTION: Enlarges the center of the current picture by a factor of 2 in each direction. This process results in a 4X loss of information as the pixels outside the center to be magnified are lost. Center pixels are replicated four times each to create the new picture.

### 3.1.8 Negative

COMMAND: i [RETURN] n [RETURN]

FUNCTION:  $p(x) = 255 - p(x)$

DESCRIPTION: Creates a reversed contrast image.

### 3.1.9 Pratt's Noise Cleaning Algorithm

COMMAND: i [RETURN] p [RETURN]

FUNCTION: Reduces noise from picture.

DESCRIPTION: Cleans noise generated by a video camera or image processing. The noise threshold is set by the input value of  $\epsilon$ . If the absolute value of the difference between a pixel value and the average of its eight neighbors is greater than  $\epsilon$ , then the pixel value is set to the average of its neighbors.

### 3.1.10 Resolution Change

COMMAND: i [RETURN] r [RETURN]

FUNCTION: Changes resolution format.

DESCRIPTION: Reduces or expands the vertical by horizontal resolution format of the current picture. In resolution reduction the pixels are averaged; this results in an information loss, and the new picture has fewer pixels. In resolution expansion the pixels are replicated with no information gain although the new picture has more pixels. If the resolution is chosen to remain the same the picture is unchanged.

### 3.1.11 Threshold

COMMAND: i [RETURN] t [RETURN]

FUNCTION: if (  $p(x) \geq \text{lower bound}$  ) and (  $p(x) \leq \text{upper bound}$  ) then  
 $p(x) = \text{new value}$

DESCRIPTION: Sets each pixel to a new value if that pixel is within the desired range. The range is determined by the upper and lower bound.

## 3.2 Display Commands

### 3.2.1 Gaussian Curve Fit

COMMAND: d [RETURN] g [RETURN]

FUNCTION: Fits a Gaussian curve to a picture.

DESCRIPTION: Calculates the centroid or first moment of a Gaussian image. A horizontal or vertical line that passes through the centroid may be selected. A Gaussian curve fit is performed using a least squares fit to the linearized data of the line.

The  $1/e^2$ , 5%, 4%, and 3% points are calculated from the parameters of the fitted Gaussian curve. Also a plot of the selected line is generated with the Gaussian curve shown as a continuous line and the actual data points represented by dots.

OPTION(S): We must enter a minimum noise cutoff value. Pixel values less than or equal to this value are ignored in calculating the centroid and Gaussian curve as these values are taken to be noise. We can also select either a horizontal or vertical fit.

### 3.2.2 Multimode Curve Fit (g-Profile)

COMMAND: d [RETURN] m [RETURN]

DESCRIPTION: Calculates the centroid of the multimode profile. A horizontal or vertical line that passes through the centroid may be selected. A g-Profile fit is performed using a least-squares fit on the data in the 10 percent to 80 percent range (EIA standard) of  $I_{\max}$  which is the maximum pixel value of the digitized image. The parameters a (core radius) and g are calculated.

OPTION(S): We must enter a minimum noise cutoff value. Pixel values less than that or equal to this value are ignored in the calculation of the centroid. We can also enter a radius of exclusion which omits all data points within a certain radius from the center.

A horizontal or vertical fit may be selected.

### 3.2.3 Intensity Histogram

COMMAND: d [RETURN] h [RETURN]

DESCRIPTION: Calculates the relative frequency versus pixel value over the entire picture and creates a graph.

OPTION(S): The output graph can be displayed in either the line or bar graph mode. If the bar graph mode is chosen the bar width may be specified.

A one-line message may be entered to be displayed on the output graph.

### 3.2.4 Line Intensity Scan (2-D)

COMMAND: d [RETURN] 2 [RETURN]

DESCRIPTION: Displays intensity as a function of position of a line across the picture.

OPTIONS: The user can use either a horizontal or vertical scan across the picture. If a horizontal scan is selected the user must input the vertical index or position of the scan. If a vertical scan is selected the user must input the horizontal index or position of the scan. The output graph can be displayed in either the line or bar graph mode.

A one-line message may be entered to be displayed on the output graph.

### 3.2.5 3-D Intensity Plot

COMMAND: `d [RETURN] 3 [RETURN]`

DESCRIPTION: Creates a three-dimensional intensity plot of the picture. The plot can have a maximum of 60 horizontal lines and 256 points per line. Since the function is limited to only 60 horizontal lines all of the information of the highest resolution pictures (128x120 or 256x240) cannot be displayed.

OPTION(S): In the highest resolution pictures (128x120 or 256x240) the user may select a 4X magnification around the center point.

Autoscaling can be used to find the lowest and highest pixels contained in the picture to ensure that the plot has the appropriate height.

We can chose between a single-line or a cross-hatch type of 3-D graph. Also a one-line message may be entered to be displayed on the output plot.

## 3.3 File Commands

### 3.3.1 Read

COMMAND: `r [RETURN]`

DESCRIPTION: Reads a picture data file into memory. To read the file the user must input the complete picture file name (and path if necessary). After the file is read the number of bytes read and resolution format of the picture are displayed. On an illegal resolution format, an error will occur (see picture file specifications, section 2.2).

### 3.3.2 Write

COMMAND: `w [RETURN]`

DESCRIPTION: Writes a picture in memory to a file. To write the file the user must input the complete picture file name (and path if necessary).



## Appendix B. Source Code

The source code for this software is made up of four executable files. The first file, IMPROC.BAT is the main batch file responsible for controlling the other three files. The second file, IMPROC#1.EXE, is an executable compiled C file of the image processing program IMPROC#1.C. The third file, IMPROC#2.BAS, is a BASIC program which creates the two-dimensional graphs. And the fourth file, IMPROC#3.EXE, is an executable compiled BASIC file of the program IMPROC#3.BAS which creates the three-dimensional plots.

There are also other necessary files which are not listed in this appendix. The file, INIT\_FFT.COM (Rapid Image Software, Tijeras, N.M.), is a machine code FFT program and must be executed before running the software. The file, GRAPHICS.COM (IBM Corp.) must also be executed before running the software for proper printing of graphics. The files, BASICA.COM and BASRUN.EXE must be present for the execution of IMPROC#2.BAS and IMPROC#3.EXE [11].

### IMPROC.BAT

```
echo off
if exist improc#1.pic erase improc#1.pic
if exist improc#2.dat erase improc#2.dat
if exist improc#3.dat erase improc#3.dat
:loop
mode mono
cls
improc#1 =32768
if exist improc#2.dat mode co80
if exist improc#2.dat basica improc#2
if exist improc#3.dat mode co80
if exist improc#3.dat improc#3
if not exist improc#1.pic goto end
goto loop
:end
mode mono
cls
```

## IMPROC#1.C

```
#include <stdio.h>          /* standard input/output routines */
#include <fcntl.h>          /* memory/hardware functions */
#include <math.h>           /* math routines */
#include <limits.h>         /* standard limits */

#define MAXLINE 40          /* # of char. read as screen input */
#define MAXMAT 50          /* #-1 of non-zero convolution elements */
#define PSIZESIZE 4        /* # of elements in psize array */
#define MAXPSIZE 61440     /* maximum picture size (60K) */
#define FBYTES 240        /* file IO block size */
#define MAXFFT 4096        /* maximum FFT array length */
#define EIAMMMAX 0.8       /* EIA multi-mode maximum point */
#define EIAMMIN 0.1        /* EIA multi-mode minimum point */

main()
{
    char line[MAXLINE],*malloc(),*p;
    /* line: array for user input */
    /* malloc(): memory allocation function */
    /* p[]: single dimension picture array */
    unsigned psize[PSIZESIZE];
    /* psize[]: picture size attributes array
       psize[0]: total picture array length
       psize[1]: vertical picture length
       psize[2]: horizontal picture length
       psize[3]: picture resolution format number (1..5) */
    int *exit,quit,ret;
    /* exit: program exit flag */

    p=malloc(MAXPSIZE); /* picture array memory allocation */
    if (p==NULL) printf("***** malloc error *****\n");
    init_pic(p,psize);
    /* reset interrupted picture array from a previous program to execute a
       graphics operation (BASIC) */

    quit=0;
    while (quit==0) {
        printf("\n");
        printf("Main Menu:\n");
        printf("\td = display functions (graphics)\n");
        printf("\ti = image processing functions\n");
        printf("\tr = read a file\n");
        printf("\tw = write a file\n");
        printf("\tq = quit\n");

        printf("\n==> "); ret=getline(line);
        if (ret!=0) {
            switch (line[0]) {
                case 'D':
                case 'd': display(p,psize,exit);
                        if (*exit==1) { save_pic(p,psize); quit=1;};
                        /* save current picture before exiting program */
            }
        }
    }
}
```

```

        break;
    case 'I':
    case 'i': image(&p,psize); break;
    case 'R':
    case 'r': read_pic(p,psize); break;
    case 'W':
    case 'w': write_pic(p,psize); break;
    case 'Q':
    case 'q': quit=1; break;
    default: printf("%c***** invalid command *****\n",7); break;
};
};
};

ret=free(p); /* free picture memory */
}

display(p,psize,exit)
char *p;
unsigned *psize;
int *exit;
{
    char line[MAXLINE];
    int ret;

    printf("\n");
    printf("Display Menu:\n");
    printf("\tg = gaussian curve fit\n");
    printf("\th = histogram of intensity graph\n");
    printf("\tm = multi-mode curve fit (g-profile)\n");
    printf("\t2 = 2-D line intensity graph\n");
    printf("\t3 = 3-D intensity plot\n");
    printf("\tq = quit\n");

    printf("\n=====> "); ret=getline(line);
    *exit=0;
    if (ret!=0) {
        switch (line[0]) {
            /* exit will end the program to execute graphics program (BASIC) */
            case 'G':
            case 'g': gaussian(p,psize); *exit=1; break;
            case 'H':
            case 'h': histogram(p,psize); *exit=1; break;
            case 'M':
            case 'm': multi_mode(p,psize); *exit=1; break;
            case '2': graph_2d(p,psize); *exit=1; break;
            case '3': plot_3d(p,psize); *exit=1; break;
            case 'Q':
            case 'q': break;
            default: printf("%c***** invalid command *****\n",7); break;
        };
    };
}
}

```

```

image(pp,psize)
char **pp;
unsigned *psize;
{
    char line[MAXLINE],*p;
    int mv[MAXMAT],ret;
    /* mv[]: convolution matrix value array */
    unsigned ma[MAXMAT];
    /* ma[]: convolution matrix relative address array */

    printf("\n");
    printf("Image Menu:\n");
    printf("\ta = average with another file\n");
    printf("\tb = bias\n");
    printf("\tc = convolve\n");
    printf("\td = demagnification\n");
    printf("\te = enhance contrast\n");
    printf("\tf = fourier transform (2-D FFT)\n");
    printf("\tm = magnification\n");
    printf("\tn = negative (reverse contrast)\n");
    printf("\tp = Pratt's noise cleaning algorithm\n");
    printf("\tr = resolution change\n");
    printf("\tt = threshold\n");
    printf("\tq = quit\n");

    printf("\n=====> "); ret=getline(line);
    if (ret!=0) {
        p=*pp;
        switch (line[0]) {
            case 'A':
            case 'a': average(p,psize); break;
            case 'B':
            case 'b': bias(p,psize); break;
            case 'C':
            case 'c': convolve(pp,psize,ma,mv,1,0); break;
            case 'D':
            case 'd': demag(pp,psize); break;
            case 'E':
            case 'e': enhance(p,psize); break;
            case 'F':
            case 'f': fourier(p,psize); break;
            case 'M':
            case 'm': mag(pp,psize); break;
            case 'N':
            case 'n': negative(p,psize); break;
            case 'P':
            case 'p': pratt(pp,psize,ma,mv); break;
            case 'R':
            case 'r': resolution(pp,psize); break;
            case 'T':
            case 't': threshold(p,psize); break;
            case 'Q':
            case 'q': break;
            default: printf("%c***** invalid command *****\n",7); break;
        }
    }
}

```

```

    );
};
)

average(p,psize)
char *p;
unsigned *psize;
{
    char *malloc(),*temp;
    int ret;
    unsigned psize0,tempsize[PSIZESIZE],u;

    temp=malloc(MAXPSIZE);
    if (temp==NULL) printf("%c***** malloc error *****\n",7);
    else {
        read_pic(temp,tempsize);
        if (tempsize[0]!=psize[0])
            printf("%cResolution of input does not match current file.",7);
        else {
            printf("Averaging with input file...\n");
            psize0=psize[0];
            for (u=0; u<psize0; ++u) p[u]=(p[u]+temp[u]+1)/2;
        };
        ret=free(temp);
    };
}

bias(p,psize)
char *p;
unsigned *psize;
{
    int biasvalue,exit,mode,peak,value;
    unsigned h256[256],psize0,u;

    printf("Select mode (0=manual,1=auto-bias) [0]: "); mode=getnum();
    if (mode!=1) mode=0;
    psize0=psize[0];

    if (mode==0) {
        exit=0;
        while (exit==0) {
            printf("Enter bias value [-255,...,255]: "); biasvalue=getnum();
            if ((biasvalue>=-255)&&(biasvalue<=255)) exit=1;
        };
    };

    if (mode==1) {
        printf("Computing Histogram...\n");
        for (u=0; u<256; ++u) h256[u]=0;
        for (u=0; u<psize0; ++u) ++h256[p[u]];
        peak=0;
        for (u=0; u<256; ++u) {
            if (h256[u]>peak) {
                peak=h256[u]; biasvalue=u;
            }
        }
    }
}

```

```

    };
};
printf("Peak value is %d\n",biasvalue); biasvalue=-biasvalue;
};

printf("Adding bias of %d...\n",biasvalue);
for (u=0; u<psize0; ++u) {
    value=p[u]+biasvalue;
    if (value<0) p[u]=0;
    else {
        if (value>255) p[u]=255;
        else p[u]=value;
    };
};
}

calc_res(psize)
unsigned *psize;
{
    switch ((int)(psize[0]/2)) {
        case 120: psize[1]= 15; psize[2]= 16; psize[3]=1; break;
        case 480: psize[1]= 30; psize[2]= 32; psize[3]=2; break;
        case 1920: psize[1]= 60; psize[2]= 64; psize[3]=3; break;
        case 7680: psize[1]=120; psize[2]=128; psize[3]=4; break;
        case 30720: psize[1]=240; psize[2]=256; psize[3]=5; break;
        default : printf("%c***** invalid file size *****\n",7);
                    psize[0]=0; psize[1]=0; psize[2]=0; psize[3]=0;
                    break;
    };
}

convolve(pp,psize,ma,mv,option,epsilon)
char **pp;
unsigned *psize,*ma;
int *mv,option,epsilon;
/* option 1 is a user inputed convolution, option 2 is for Pratt's algorithm */
/* epsilon: threshold value from Pratt's algorithm */
{
    char *malloc(),*p,*temp;
    int bias,diff,i,n,norm,sum;
    unsigned psize0,psize2,u,v;

    p=*pp; temp=malloc(MAXPSIZE);
    if (temp==NULL) printf("***** malloc error *****\n");
    else {
        psize0=psize[0]; psize2=psize[2];

        bias=0;
        if (option==1) {
            input_matrix(psize,ma,mv);
            printf("\n");
            printf("Enter bias value [0]: "); bias=getnum();
        };
        n=ma[0]; norm=0;

```

```

for (i=1; i<=n; ++i) norm+=mv[i];
if (norm==0) norm=1;
if (option==1) {
    printf("Enter normalization value [%d]: ",norm); i=getnum();
    if (i!=0) norm=i;
};
printf("Bias = %d    Normalization = %d\n",bias,norm);

printf(" 0%% completed\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b");
u=0;
while (u<psize0) {
    for (v=0; v<psize2; ++v) {
        sum=0;
        for (i=1; i<=n; ++i)
            if (u+ma[i]<psize0) sum+=mv[i]*(int)p[u+ma[i]];
        sum=sum/norm;
        if (option==1) {
            sum+=bias;
            if (sum<0) temp[u]=0;
            else if (sum>255) temp[u]=255;
            else temp[u]=sum;
        }
        else {
            diff=sum-(int)p[u];
            if (diff<0) diff=(-diff);
            if (diff<epsilon) temp[u]=p[u];
            else temp[u]=sum;
        };
        ++u;
    };
    printf("%3.0f\b\b\b\b", (100.0*u)/psize0);
};
printf("\n");
*pp=temp; i=free(p);
};
}

demag(pp,psize)
char **pp;
unsigned *psize;
{
    char *malloc(),*p,*temp;
    int ret,sum;
    unsigned psize0,psize1,psize2,px,py,tempx,tempxstart,tempxfinish,
        tempy,tempystart,tempyfinish,u;

    printf("Demagnifying by 2X...\n");
    p=*pp; temp=malloc(MAXPSIZE);
    if (temp==NULL) printf("%c***** malloc error *****\n",7);
    else {
        psize0=psize[0]; psize1=psize[1]; psize2=psize[2];
        for (u=0; u<psize0; ++u) temp[u]=0;

        tempxstart=psize2/4; tempxfinish=(3*psize2)/4;

```

```

    tempystart=(psize1/4)*psize2; tempyfinish=((3*psize1)/4)*psize2;
    py=0;
    for (tempy=tempystart; tempy<tempyfinish; tempy+=psize2) {
        px=0;
        for (tempx=tempxstart; tempx<tempxfinish; ++tempx) {
            sum=p[px+py]+p[px+1+py]+p[px+py+psize2]+p[px+1+py+psize2];
            temp[tempx+tempy]=(sum+2)/4;
            px+=2;
        };
        py+=(2*psize2);
    };

    *pp=temp; ret=free(p);
};
}

enhance(p,psize)
char *p;
unsigned *psize;
{
    int exit,low,high,mode;
    unsigned psize0,u;
    float slope,value;

    printf("Select mode (0=manual,1=auto-enhance) [0]: "); mode=getnum();
    if (mode!=1) mode=0;

    psize0=psize[0];
    if (mode==0) {
        exit=0;
        while (exit==0) {
            printf("Enter lowest value (0,...,254): "); low=getnum();
            if ((low>=0)&&(low<=254)) exit=1;
        };
        exit=0;
        while (exit==0) {
            printf("Enter highest value (%d,...,255): ",low+1);
            high=getnum();
            if ((high>=(low+1))&&(high<=255)) exit=1;
        };
    };
    if (mode==1) {
        low=255; high=0;
        for (u=0; u<psize0; ++u) {
            if (p[u]<low) low=p[u];
            if (p[u]>high) high=p[u];
        };
        printf("Lowest value is %d, Highest value is %d.\n",low,high);
    };

    printf("Enhancing contrast...\n");
    slope=255.0/(high-low);
    for (u=0; u<psize0; ++u) {
        value=slope*(p[u]-low);

```

[illegible]

[illegible]

[illegible]

```

        sumyp+=py*c;
    };
};
centerx=((float)sumxp)/sump;
centery=((float)sumyp)/sump;
printf("Centroid is (%.2f,%.2f)\n",centerx,centery);

exit=0;
while (exit==0) {
    printf("Select horizontal or vertical fit (h,v): ");
    ret=getline(line);
    if (ret!=0) {
        switch (line[0]) {
            case 'H':
            case 'h': loop=psize[2]; offset=1; option=3;
                    center=centerx; index=centery+0.5;
                    exit=1; break;

            case 'V':
            case 'v': loop=psize[1]; offset=psize[2]; option=4;
                    center=centery; index=centerx+0.5;
                    exit=1; break;

            default: printf("%c***** invalid option *****\n",7); break;
        };
    };
};

printf("Fitting to a Gaussian...\n");
file=creat("improc#2.dat",-1);
ret=write(file,&option,1); ret=write(file,&index,1);
c=loop-1; ret=write(file,&c,1);
if (option==3) index*=psize[2];

sumx=0; sumx2=0; sumy=0; sumxy=0; k=0;
for (u=0; u<loop; ++u) {
    c=p[(unsigned)u*offset+index];
    if (c>cutoff) {
        ++k;
        x=(u-center)*(u-center);
        y=log((float)c);
        sumx+=x;
        sumx2+=x*x;
        sumy+=y;
        sumxy+=x*y;
    };
};

m=(k*sumxy-sumx*sumy)/(k*sumx2-sumx*sumx);
b=(sumx2*sumy-sumx*sumxy)/(k*sumx2-sumx*sumx);
alpha=-m;
a=exp(b);
printf("A = %f   alpha = %e   c = %f\n",a,alpha,center);
printf("Beam spot radius:\n");
printf("\t1 over e squared = %f\n",sqrt(2.0/alpha));
printf("\t5 percent = %f\n",sqrt(2.99573/alpha));

```

```

printf("\t4 percent = %f\n",sqrt(3.21888/alpha));
printf("\t3 percent = %f\n",sqrt(3.50656/alpha));
printf("Press RETURN to get graph.\n"); ret=getline(line);

printf("Watch graphics monitor...\n");
for (u=0; u<loop; ++u) {
    value=a*exp(-alpha*(u-center)*(u-center));
    if (value>255.0) c=255; else c=value+0.5;
    if (c==26) c=25;
    /* Note: this step is taken because character 26 is EOF in BASIC which
       halts the reading of a file (ie. 26 is illegal) */
    ret=write(file,&c,1);
};
for (u=0; u<loop; ++u) {
    c=p[(unsigned)u*offset+index];
    if (c==26) c=25;
    /* Note: this step is taken because character 26 is EOF in BASIC which
       halts the reading of a file (ie. 26 is illegal) */
    ret=write(file,&c,1);
};
ret=close(file);
}

getline(s)
char s[];
{
    int c,i;

    for (i=0; i<MAXLINE-1 && (c=getchar())!=EOF && c!='\n'; ++i)
        s[i]=c;
    s[i]='\0'; return(i);
}

getnum()
{
    char line[MAXLINE];
    int i,n;

    i=getline(line); i=sscanf(line,"%d",&n);
    if (i==1) return(n); else return(0);
}

graph_2d(p,psize)
char *p;
unsigned *psize;
{
    char c,line[MAXLINE];
    int exit,i,file,lim,option,ret;
    unsigned index,loop,off,x;

    printf("2-D line intensity graph\n");
    exit=0;
    while (exit==0) {
        printf("Select horizontal or vertical scan (h,v): ");

```

```

ret=getline(line);
if (ret!=0) {
    switch (line[0]) {
        case 'H': case 'h':
            lim=psize[1]-1; loop=psize[2]; off=1; option=0;
            exit=1; break;
        case 'V': case 'v':
            lim=psize[2]-1; loop=psize[1]; off=psize[2]; option=1;
            exit=1; break;
        default: printf("%c***** invalid option *****\n",7); break;
    };
};
exit=0;
while (exit==0) {
    printf("Enter index value (0,1,...,%d): ",lim); index=getnum();
    if ((index>=0)&&(index<=lim)) exit=1;
    else printf("%c***** illegal value *****\n",7);
};
printf("watch graphics monitor...\n");

file=creat("improc#2.dat",-1);
ret=write(file,&option,1); ret=write(file,&index,1);
c=loop-1; ret=write(file,&c,1);
if (option==0) index*=psize[2];
for (x=0; x<loop; ++x) {
    c=p[(unsigned)x*off+index];
    if (c==26) c=25;
    /* Note: this step is taken because character 26 is EOF in BASIC which
       halts the reading of a file (ie. 26 is illegal) */
    ret=write(file,&c,1);
};
ret=close(file);
}

histogram(p,psize)
char *p;
unsigned *psize;
{
    int ave,file,i,j,highval,lowval,option,peakval,ret;
    unsigned u,peak,h256[256];
    long sum;

    printf("Computing histogram...\n");
    for (i=0; i<256; ++i) h256[i]=0;
    for (u=0; u<psize[0]; ++u) ++h256[p[u]];
    sum=0;
    for (i=0; i<256; ++i) sum+=(long)i*h256[i];
    peak=0;
    for (i=0; i<256; ++i) if (h256[i]>peak) { peak=h256[i]; peakval=i; };
    highval=255;
    while (h256[highval]==0) --highval;
    lowval=0;
    while (h256[lowval]==0) ++lowval;
}

```

```

ave=(int)((float)sum/psize[0]);
printf("Creating histogram graph (watch graphics monitor)...\n");
file=creat("improc#2.dat",-1);
option=2; ret=write(file,&option,1);
ret=write(file,&lowval,1); ret=write(file,&highval,1);
ret=write(file,&ave,1); ret=write(file,&peakval,1);
for (i=0; i<256; ++i) {
    j=((long)255*h256[i])/peak;
    if (j==0 && h256[i]>0) j=1;
    if (j==26) j=25;
    /* Note: this step is taken because character 26 is EOF in BASIC which
       halts the reading of a file (ie. 26 is illegal) */
    ret=write(file,&j,1);
};
ret=close(file);
}

init_pic(p,psize)
char *p;
unsigned *psize;
{
    int i,file,ret;

    file=open("improc#1.pic",O_RDONLY | 0x8000);
    if (file==-1) {
        /* interrupted picture file does not exist, begin fresh */
        printf("\nIMAGE PROCESSING PROGRAM\n");
        printf("by Matt Weppner (NBS 1986)\n");
        for (i=0; i<PSIZESIZE; ++i) psize[i]=0;
    }
    else {
        /* interrupted picture file is retrieved */
        psize[0]=0; ret=FBYTES;
        while (psize[0]<MAXPSIZE && ret==FBYTES) {
            ret=read(file,p,FBYTES); p+=ret; psize[0]+=ret;
        };
        calc_res(psize);
    };
    ret=close(file); ret=unlink("improc#1.pic");
}

input_matrix(psize,ma,mv)
unsigned *psize;
int *ma,*mv;
{
    int a,i,j,n,v,x,y;

    printf("Convolution with input matrix\n");
    printf("Enter matrix height (1,3,...): "); y=getnum();
    printf("Enter matrix width (1,3,...): "); x=getnum();
    printf("\nRow\tColumn\tValue\n");
    n=0;
    for (j=1; j<=y; ++j) {
        for (i=1; i<=x; ++i) {

```

```

        printf("%d\t%d\t",j,i); v=getnum();
        if (v!=0) {
            ++n;
            ma[n]=psize[2]*(j-(y/2)-1)+(i-(x/2)-1); mv[n]=v;
        };
    };
};
printf("\n%d non-zero elements.\n",n); ma[0]=n;
}

mag(pp,psize)
char **pp;
unsigned *psize;
{
    char c,*malloc(),*p,*temp;
    int ret;
    unsigned psize1,psize2,px,pxstart,pxfinish,py,pystart,pyfinish,
        tempx,tempy;

    printf("Magnifying by 2X...\n");
    p=*pp; temp=malloc(MAXPSIZE);
    if (temp==NULL) printf("%c***** malloc error *****\n",7);
    else {
        psize1=psize[1]; psize2=psize[2];
        pxstart=psize2/4; pxfinish=(3*psize2)/4;
        pystart=(psize1/4)*psize2; pyfinish=((3*psize1)/4)*psize2;
        tempy=0;
        for (py=pystart; py<pyfinish; py+=psize2) {
            tempx=0;
            for (px=pxstart; px<pxfinish; ++px) {
                c=p[px+py];
                temp[tempx+tempy]=c; temp[tempx+1+tempy]=c;
                temp[tempx+tempy+psize2]=c; temp[tempx+1+tempy+psize2]=c;
                tempx+=2;
            };
            tempy+=(2*psize2);
        };

        *pp=temp; ret=free(p);
    };
}

multi_mode(p,psize)
char *p;
unsigned *psize;
{
    char c,line[MAXLINE],pmax,pmin;
    int cutoff,exit,file,i0,imax,k,n,option,ret,radius,step;
    unsigned index,loop,offset,psize1,psize2,px,py,u;
    float a,b,center,centerx,centery,g,e,m,r,sumx,sumy,sumxy,sumx2,value,x,y;
    float olda,olde,oldg,rarray[256],xarray[256],yarray[256];
    long sump,sumxp,sumyp;

    printf("Multi-mode Curve Fit:\n");

```

```

exit=0;
while (exit==0) {
    printf("Enter maximum pixel noise cutoff value (0,1,...) [0]: ");
    cutoff=getnum();
    if ((cutoff>=0)&&(cutoff<=255)) exit=1;
    else printf("%c***** invalid value *****\n",7);
};

printf("Calculating centroid...\n");
sump=0; sumxp=0; sumyp=0;
u=0; psize1=psize[1]; psize2=psize[2];
for (py=0; py<psize1; ++py) {
    for (px=0; px<psize2; ++px) {
        c=p[u]; ++u;
        if (c>cutoff) {
            sump+=c;
            sumxp+=px*c;
            sumyp+=py*c;
        };
    };
};
centerx=((float)sumxp)/sump;
centery=((float)sumyp)/sump;
printf("Centroid is (%.2f,%.2f)\n",centerx,centery);

exit=0;
while (exit==0) {
    printf("Select horizontal or vertical fit (h,v): ");
    ret=getline(line);
    if (ret!=0) {
        switch (line[0]) {
            case 'H':
                case 'h': loop=psize[2]; offset=1; option=5;
                    center=centerx; index=centery+0.5;
                    exit=1; break;
            case 'V':
                case 'v': loop=psize[1]; offset=psize[2]; option=6;
                    center=centery; index=centerx+0.5;
                    exit=1; break;
            default: printf("%c***** invalid option *****\n",7); break;
        };
    };
};

exit=0;
while (exit==0) {
    printf("Enter radius of exclusion (0,1,...) [0]: "); radius=getnum();
    if ((radius>=0)&&(radius<=loop/3)) exit=1;
    else printf("%c***** invalid value *****\n",7);
};

printf("Fitting to a g-profile...\n");
file=creat("improc#2.dat",-1);

```

```

ret=write(file,&option,1); ret=write(file,&index,1);
c=loop-1; ret=write(file,&c,1);
if (option==5) index*=psize[2];

imax=0;
for (u=0; u<loop; ++u) {
    c=p[(unsigned)u*offset+index];
    if (c>imax) imax=c;
};
printf("imax = %d\n",imax);

pmax=imax*EIAMMMAX; pmin=imax*EIAMMMIN;
k=0; sumx=0; sumx2=0;
for (u=0; u<loop; ++u) {
    c=p[(unsigned)u*offset+index]; r=fabs(u-center);
    if ((c>=pmin)&&(c<=pmax)&&(r>radius)) {
        rarray[k]=r; x=log(r); xarray[k]=x; yarray[k]=c;
        sumx+=x ; sumx2+=x*x; ++k;
    };
};
printf("%d data points used.\n",k);

printf("Enter starting value for Io [%d]: ",imax); i0=getnum();
if ((i0==0)|| (i0>=255)) i0=imax;
n=0; i0+=1; step=-1;
e=1.0e10; a=0; g=0;
while (((e<olde)|| (n<=2))&&(i0>=0)&&(i0<=255)) {
    olde=e; olda=a; oldg=g;
    ++n; i0+=step;
    sumy=0; sumxy=0;
    for (u=0; u<k; ++u) {
        x=xarray[u]; y=log(1.0-yarray[u]/i0);
        sumy+=y; sumxy+=x*y;
    };
    m=(k*sumxy-sumx*sumy)/(k*sumx2-sumx*sumx); g=m;
    b=(sumx2*sumy-sumx*sumxy)/(k*sumx2-sumx*sumx); a=exp(-b/m);
    e=0;
    for (u=0; u<k; ++u) {
        value=yarray[u]-i0*(1.0-exp(g*log(rarray[u]/a)));
        e+=value*value;
    };
    printf("n=%d Io=%d error=%.4e\n",n,i0,e);
    if (e>olde) step=-step;
};
i0+=step; a=olda; g=oldg;
printf("\nFinal Parameters: Io = %d a = %f (pixels) g = %f\n\n",i0,a,g);
printf("Press RETURN to get graph.\n"); ret=getline(line);

printf("Watch graphics monitor...\n");
for (u=0; u<loop; ++u) {
    value=i0*(1.0-exp(g*log(fabs(u-center)/a)));
    if (value>255.0) c=255;
    else { if (value<0.0) c=0; else c=value+0.5; };
    if (c==26) c=25;
};

```

```

    /* Note: this step is taken because character 26 is EOF in BASIC which
       halts the reading of a file (ie. 26 is illegal) */
    ret=write(file,&c,1);
};

for (u=0; u<loop; ++u) {
    c=p[(unsigned)u*offset+index];
    if (c==26) c=25;
    /* Note: this step is taken because character 26 is EOF in BASIC which
       halts the reading of a file (ie. 26 is illegal) */
    ret=write(file,&c,1);
};

ret=close(file);
}

negative(p,psize)
char *p;
unsigned *psize;
{
    unsigned psize0,u;

    printf("Creating negative picture (reversing contrast)...\n");
    psize0=psize[0];
    for (u=0; u<psize0; ++u) p[u]=255-p[u];
}

plot_3d(p,psize)
char *p;
unsigned *psize;
{
    char c,lowest,highest,line[MAXLINE],*temp,*malloc();
    int exit,file,i,ret,zoom;
    unsigned center,tempsize,t,u,v,
        x,y,xsize,ysize,xcenter,ycenter,xmin,ymin,xmax,ymax,yjump;

    printf("3-D intensity graph\n");
    zoom=0;
    if (psize[3]>=4) {
        exit=0;
        while (exit==0) {
            printf("Select full size or 4X enlargement (f,e): ");
            ret=getline(line);
            if (ret!=0) {
                switch (line[0]) {
                    case 'F':
                    case 'f': exit=1; break;
                    case 'E':
                    case 'e': zoom=1; exit=1; break;
                    default: printf("%c***** invalid entry *****",7); break;
                }
            }
        }
    };
};
};
};

```

```

if (zoom==1) {
    xmin=psize[2]/8-1; xmax=psize[2]-(xmin+2); center=psize[2]/2-1;
    printf("Enter center horizontal position ");
    printf("(%d,...,%d) [%d]: ",xmin,xmax,center);
    xcenter=getnum();
    if ((xcenter<xmin)|| (xcenter>xmax)) xcenter=center;
    ymin=psize[1]/8-1; ymax=psize[1]-(ymin+2); center=psize[1]/2-1;
    printf("Enter center vertical position ");
    printf("(%d,...,%d) [%d]: ",ymin,ymax,center);
    ycenter=getnum();
    if ((ycenter<ymin)|| (ycenter>ymax)) ycenter=center;
};
printf("Creating 3-D plot (watch graphics monitor)...\n");

switch ((int)psize[3]) {
    case 1: ysize=15; xsize=16; break;
    case 2: ysize=30; xsize=32; break;
    case 3: ysize=60; xsize=64; break;
    case 4: if (zoom==0) {ysize=60; xsize=128; }
            else { ysize=30; xsize=32; };
            break;
    case 5: ysize=60;
            if (zoom==0) { xsize=256; } else { xsize=64; };
            break;
};
yjump=(psize[1]/ysize)*psize[2];
tempsize=ysize*xsize; temp=malloc(tempsize);
if (temp==NULL) printf("***** malloc error *****\n");
file=creat("improc#3.dat",-1);
c=ysize-1; ret=write(file,&c,1);
c=xsize-1; ret=write(file,&c,1);
ret=write(file,&zoom,1);
lowest=255; highest=0;

if (zoom==0) {
    t=0; u=psize[0]-psize[2];
    for (y=0; y<ysize; ++y) {
        for (x=0; x<xsize; ++x) {
            c=p[u+x]; if (c==26) c=25; temp[t]=c; ++t;
            /* Note: this step is taken because character 26 is EOF in BASIC
               which halts the reading of a file (ie. 26 is illegal) */
            if (c<lowest) lowest=c; if (c>highest) highest=c;
        };
        u-=yjump;
    };
    ret=write(file,&lowest,1); ret=write(file,&highest,1);
    ret=write(file,temp,tempsize);
}
else {
    t=0; u=psize[2]*(ycenter+psize[1]/8); xcenter-=psize[2]/8-1;
    for (i=0; i<ysize; ++i) {
        for (v=0; v<xsize; ++v) {
            c=p[u+v+xcenter]; if (c==26) c=25; temp[t+v]=c;
            /* Note: this step is taken because character 26 is EOF in BASIC

```

```

        which halts the reading of a file (ie. 26 is illegal) */
        if (c<lowest) lowest=c; if (c>highest) highest=c;
    };
    t+=xsize; u-=psize[2];
};
ret=write(file,&lowest,1); ret=write(file,&highest,1);
ret=write(file,temp,tempsize);
};
ret=close(file); ret=free(temp);
}

pratt(pp,psize,ma,mv)
char **pp;
unsigned *psize;
int *ma,*mv;
{
    int i,epsilon;

    printf("Pratt's noise cleaning algorithm\nEnter epsilon: ");
    epsilon=getnum();
    ma[0]=8;
    i=psize[2];
    ma[1]=-(i+1); ma[2]=-i; ma[3]=-(i-1); ma[4]=-1;
    ma[5]=1; ma[6]=i-1; ma[7]=i; ma[8]=i+1;
    for (i=1; i<=8; ++i) mv[i]=1;
    convolve(pp,psize,ma,mv,2,epsilon);
}

read_pic(p,psize)
char *p;
unsigned *psize;
{
    char line[MAXLINE];
    int file,ret;

    printf("Enter input filename: "); ret=getline(line);
    printf("Reading file %s\n",line);
    file=open(line,O_RDONLY | 0x8000);
    if (file==-1) printf("%c***** cannot open file *****\n",7);
    else {
        psize[0]=0; ret=FBYTES;
        while (psize[0]<MAXPSIZE && ret==FBYTES) {
            ret=read(file,p,FBYTES); p+=ret; psize[0]+=ret;
        };
        printf("%u bytes read.\n",psize[0]);
        calc_res(psize);
        printf("Resolution is %ux%u\n",psize[2],psize[1]);
    };
    ret=close(file);
}

resolution(pp,psize)
char **pp;
unsigned *psize;

```

```

{
char *malloc(),*p,*temp;
int exit,i,ret;
unsigned area,ratio,sum,tempsize[PSIZESIZE],u,x,xoff,y,yoff,
        psize0,psize1,psize2,tsize0,tsize1,tsize2;

printf("Resolution is %ux%u\n",psize[2],psize[1]);
exit=0;
while (exit==0) {
    printf("Enter new resolution ");
    printf("(1=16x15,2=32x30,3=64x60,4=128x120,5=256x240): ");
    ret=getnum();
    switch (ret) {
        case 1: tempsize[0]= 240; exit=1; break;
        case 2: tempsize[0]= 960; exit=1; break;
        case 3: tempsize[0]= 3840; exit=1; break;
        case 4: tempsize[0]=15360; exit=1; break;
        case 5: tempsize[0]=61440; exit=1; break;
        default: printf("%c***** invalid option *****",7); break;
    };
};
calc_res(tempsize);
if (tempsize[3]!=psize[3]) {
    p=*pp; temp=malloc(MAXPSIZE);
    if (temp==NULL) printf("%c***** malloc error *****\n",7);
    psize0=psize[0]; psize1=psize[1]; psize2=psize[2];
    tsize0=tempsize[0]; tsize1=tempsize[1]; tsize2=tempsize[2];
    printf("Changing resolution from %ux%u ",psize[2],psize[1]);
    printf("to %ux%u...\n",tempsize[2],tempsize[1]);

    if (tsize0<psize0) {
        ratio=psize1/tsize1;
        area=ratio*ratio;
        y=0; x=0;
        for (u=0; u<tsize0; ++u) {
            sum=0;
            for (yoff=0; yoff<(ratio*psize2); yoff+=psize2) {
                for (xoff=0; xoff<ratio; ++xoff) {
                    sum+=p[y+yoff+x*xoff];
                };
            };
            temp[u]=sum/area;
            x+=ratio;
            if (x==psize2) { x=0; y+=ratio*psize2; };
        };
    };
    if (tsize0>psize0) {
        ratio=tsize1/psize1;
        y=0; x=0;
        for (u=0; u<psize0; ++u) {
            for (yoff=0; yoff<(ratio*tsize2); yoff+=tsize2) {
                for (xoff=0; xoff<ratio; ++xoff) {
                    temp[y+yoff+x*xoff]=p[u];
                };
            };
        };
    };
};

```

```

        };
        x+=ratio;
        if (x==tsize2) {x=0; y+=ratio*tsize2; };
    };
};
*pp=temp; ret=free(p);
for (i=0; i<PSIZESIZE; ++i) psize[i]=tempsize[i];
};
printf("Resolution is %ux%u\n",psize[2],psize[1]);
)

save_pic(p,psize)
char *p;
unsigned *psize;
{
    int file,ret;
    unsigned sum;

    file=creat("improc#1.pic",-1);
    sum=0; ret=FBYTES;
    while (sum<psize[0] && ret==FBYTES) {
        ret=write(file,p,FBYTES); p+=ret; sum+=ret;
    };
    ret=close(file);
}

threshold(p,psize)
char *p;
unsigned *psize;
{
    int exit,low,high,new;
    unsigned psize0,u;

    exit=0;
    while (exit==0) {
        printf("Enter lower bound [0,...,255]: "); low=getnum();
        if ((low>=0)&&(low<=255)) exit=1;
    };
    exit=0;
    while (exit==0) {
        printf("Enter upper bound [%d,...,255]: ",low); high=getnum();
        if ((high>=low)&&(high<=255)) exit=1;
    };
    exit=0;
    while (exit==0) {
        printf("Enter new value [0,...,255]: "); new=getnum();
        if ((new>=0)&&(new<=255)) exit=1;
    };

    printf("Setting threshold values...\n");
    psize0=psize[0];
    for (u=0; u<=psize0; ++u)
        if ((p[u]>=low)&&(p[u]<=high)) p[u]=new;
}

```

```

write_pic(p,psize)
char *p;
unsigned *psize;
{
    char line[MAXLINE];
    int file,ret;
    unsigned sum;

    printf("Enter output filename: "); ret=getline(line);
    printf("Writing file %s\n",line);
    file=creat(line,-1);
    if (file==-1) printf("%c***** cannot open file*****\n",7);
    else {
        sum=0; ret=FBYTES;
        while (sum<psize[0] && ret==FBYTES) {
            ret=write(file,p,FBYTES); p+=ret; sum+=ret;
        };
    };
    printf("%u bytes written.\n",sum);
    ret=close(file);
}

```

## IMPROC#2.BAS

```
10 SCREEN 2:CLS
20 KEY OFF
30 OPEN "improc#2.dat" FOR INPUT AS 1
40 XSCALE=2: XB=76
50 YSCALE=.5: YB=20
60 OPT=ASC(INPUT$(1,#1))
70 PRINT "2-D PLOTTING PROGRAM"
80 PRINT
90 PRINT "When plot is completed press Shift-Prt to print hardcopy,"
100 PRINT "or any other key to exit."
110 PRINT
120 MODE=0
130 IF (OPT>=3)AND(OPT<=6) GOTO 160
140 INPUT "Select Graph Mode (0=line,1=bar)[0]:",MODE
150 PRINT
160 IF MODE<>1 THEN MODE=0
170 IF (MODE=0)OR(OPT<>2) THEN GOTO 230
180 INPUT "Enter Bar Width (odd # of pixels only) [1]:",BARWIDTH
190 PRINT
200 BARWIDTH=CINT(BARWIDTH)
210 IF BARWIDTH<1 THEN BARWIDTH=1
220 BARWIDTH=INT(BARWIDTH/2)
230 INPUT "Message:",MESSAGE$
240 CLS
250 IF OPT<>0 THEN GOTO 270
260 PRINT"Horizontal Intensity Graph: ";:XLIM=255:GOTO 450
270 IF OPT<>1 THEN GOTO 290
280 PRINT"Vertical Intensity Graph: ";:XLIM=239:GOTO 450
290 IF OPT<>2 THEN GOTO 360
300 LOW=ASC(INPUT$(1,#1)):HIGH=ASC(INPUT$(1,#1))
310 AVE=ASC(INPUT$(1,#1)):PEAK=ASC(INPUT$(1,#1))
320 XLIM=255:LOOP=255
330 PRINT "Intensity Histogram: Low=";LOW;" High=";HIGH;
340 PRINT " Average=";AVE;" Peak=";PEAK
350 GOTO 480
360 IF OPT<>3 THEN 380
370 PRINT"Horizontal Gaussian Curve Fit: ";:XLIM=255:GOTO 450
380 IF OPT<>4 THEN GOTO 400
390 PRINT"Vertical Gaussian Curve Fit: ";:XLIM=239:GOTO 450
400 IF OPT<>5 THEN GOTO 420
410 PRINT"Horizontal G-Profile Fit: ";:XLIM=255:GOTO 450
420 IF OPT<>6 THEN GOTO 440
430 PRINT"Vertical G-Profile Fit: ";:XLIM=239:GOTO 450
440 PRINT "Illegal input file":STOP
450 INDEX=ASC(INPUT$(1,#1))
460 LOOP=ASC(INPUT$(1,#1))
470 PRINT "Line Index =";INDEX
480 PRINT
490 IF OPT=2 THEN GOTO 550
500 FOR Y=256 TO 0 STEP -64
510     PRINT TAB(6);Y
520     PRINT:PRINT:PRINT
```

```

530 NEXT Y
540 GOTO 600
550 PRINT TAB(5)"1.00":PRINT:PRINT:PRINT
560 PRINT TAB(5)"0.75":PRINT:PRINT:PRINT
570 PRINT TAB(5)"0.50":PRINT:PRINT:PRINT
580 PRINT TAB(5)"0.25":PRINT:PRINT:PRINT
590 PRINT TAB(5)"0.00"
600 LINE (XB,YB)-(XB+2*XLIM+2,YB)
610 LINE (XB+2*XLIM+2,YB)-(XB+2*XLIM+2,YB+128)
620 LINE (XB,128+YB)-(XB+2*XLIM+2,128+YB)
630 LINE (XB,YB)-(XB,YB+128)
640 SSIZE=16:TSIZE=1
650 FOR X=0 TO (2*XLIM)+2 STEP SSIZE
660     LINE(XB+X,YB)-(XB+X,YB-TSIZE)
670     LINE(XB+X,YB+128)-(XB+X,YB+128+TSIZE)
680 NEXT X
690 SSIZE=8:TSIZE=3
700 FOR Y=0 TO 128 STEP SSIZE
710     LINE(XB,YB+128-Y)-(XB-TSIZE,YB+128-Y)
720     LINE(XB+2*XLIM+2,YB+128-Y)-(XB+2*XLIM+2+TSIZE,YB+128-Y)
730 NEXT Y
740 PRINT CHR$(11)
750 IF (OPT=0)OR(OPT=1)OR((OPT>=3)AND(OPT<=6)) THEN Y$="INTENSITY":Z1=5:Z2=4
760 IF (OPT=2) THEN Y$="REL FREQUENCY":Z1=3:Z2=2
770 FOR Y=1 TO Z1:PRINT:NEXT Y
780 FOR Y=1 TO LEN(Y$):PRINT MID$(Y$,Y,1):NEXT Y
790 FOR Y=1 TO Z2:PRINT:NEXT Y
800 IF (LOOP=255) THEN PRINT TAB(10)
      "0"           64           128           192           256";
810 IF (LOOP=127) THEN PRINT TAB(10)
      "0"           32           64           96           128";
820 IF (LOOP=63) THEN PRINT TAB(10)
      "0"           16           32           48           64";
830 IF (LOOP=31) THEN PRINT TAB(10)
      "0"           8           16           24           32";
840 IF (LOOP=15) THEN PRINT TAB(10)
      "0"           4           8           12           16";
850 IF (LOOP=239) THEN PRINT TAB(10)
      "0"          40          80          120          160          200          240";
860 IF (LOOP=119) THEN PRINT TAB(10)
      "0"          20          40          60          80          100          120";
870 IF (LOOP=59) THEN PRINT TAB(10)
      "0"          10          20          30          40          50          60";
880 IF (LOOP=29) THEN PRINT TAB(10)
      "0"           5          10          15          20          25          30";
890 IF (LOOP=14) THEN PRINT TAB(10)
      "0"           5           10          15          20          25          30";
900 PRINT
910 IF (OPT=0)OR(OPT=3)OR(OPT=5) THEN PRINT TAB(35);"P O S I T I O N"
920 IF (OPT=1)OR(OPT=4)OR(OPT=6) THEN PRINT TAB(33);"P O S I T I O N"
930 IF (OPT=2) THEN PRINT TAB(34);"I N T E N S I T Y"
940 PRINT:PRINT MESSAGE$;
950 FIRST=1
960 IF (OPT=0)OR(OPT=3)OR(OPT=5) THEN XSCALE=512/(LOOP+1)

```

```

970 IF (OPT=1)OR(OPT=4)OR(OPT=6) THEN XSCALE=480/(LOOP+1)
980 FOR X=0 TO LOOP
990   Y=ASC(INPUT$(1,#1))*YSCALE
1000   XSCREEN=XSCALE*X+XB:YSCREEN=128-Y+YB-.1
1010   IF MODE=1 THEN GOTO 1040
1020   IF FIRST=0 THEN LINE -(XSCREEN,YSCREEN):GOTO 1080
1030   PSET(XSCREEN,YSCREEN):FIRST=0:GOTO 1080
1040   IF (OPT<>2) THEN LINE (XSCREEN,YB+128)-(XSCREEN,YSCREEN):GOTO 1080
1050   FOR I=-BARWIDTH TO BARWIDTH
1060     LINE(XSCREEN+I,YB+128)-(XSCREEN+I,YSCREEN)
1070   NEXT I
1080 NEXT X
1090 IF NOT((OPT>=3)AND(OPT<=6)) THEN GOTO 1220
1100 I=1
1110 FIRST=1
1120 FOR X=0 TO LOOP
1130   Y=ASC(INPUT$(1,#1))*YSCALE
1140   XSCREEN=XSCALE*X+XB:YSCREEN=128-Y+YB-.1
1150   IF FIRST=1 THEN FIRST=0:PSET(XSCREEN,YSCREEN):GOTO 1210
1160   IF (LOOP>100) THEN LINE-(XSCREEN,YSCREEN):GOTO 1210
1170   LINE(XSCREEN-2*I,YSCREEN+I)-(XSCREEN+2*I,YSCREEN+I)
1180   LINE(XSCREEN-2*I,YSCREEN-I)-(XSCREEN+2*I,YSCREEN-I)
1190   LINE(XSCREEN+2*I,YSCREEN-I)-(XSCREEN+2*I,YSCREEN+I)
1200   LINE(XSCREEN-2*I,YSCREEN-I)-(XSCREEN-2*I,YSCREEN+I)
1210 NEXT X
1220 CLOSE 1
1221 BEEP
1230 A$=INKEY$:IF A$="" THEN 1230
1240 KILL "improc#2.dat"
1250 SYSTEM

```

IMPROC#3.BAS

```
10 SCREEN 2:CLS
20 KEY OFF
30 PRINT "3-D PLOTTING PROGRAM"
40 PRINT
50 PRINT "When plot is completed press Shift-PrtSc to print a hardcopy,"
60 PRINT "or any other key to exit."
70 PRINT
80 INPUT "Select Autoscale (1=on,0=off) [1]: ",AUTOSCALE$
90 PRINT
100 IF AUTOSCALE$="0" THEN AUTOSCALE=0 ELSE AUTOSCALE=1
110 INPUT "Select Graph Type (0=lines,1=cross-hatch) [0]: ",TYPE$
120 PRINT
130 IF TYPE$="1" THEN TYPE=1 ELSE TYPE=0
140 INPUT "Message: ",MESSAGE$
150 CLS
160 PRINT "THREE DIMENSIONAL INTENSITY PLOT: ";MESSAGE$
170 DIM BIGGEST(639),SMALLEST(639),TEMPBIGGEST(639),TEMPSMALLEST(639)
180 OPEN "improc#3.dat" FOR INPUT AS 1
190 YSIZE=ASC(INPUT$(1,#1)):XSIZE=ASC(INPUT$(1,#1))
200 DIM OLDLINE(256,2)
210 ZOOM=ASC(INPUT$(1,#1))
220 LOWEST=ASC(INPUT$(1,#1)):HIGHEST=ASC(INPUT$(1,#1))
230 IF AUTOSCALE=0 THEN LOWEST=0: HIGHEST=255
240 MSCALE=255/(HIGHEST-LOWEST):BSCALE=(255*LOWEST)/(LOWEST-HIGHEST)
250 HEIGHT=.27:FIRSTX=0:LASTX=XSIZE
260 FOR ELEMENT=0 TO 639
270     BIGGEST(ELEMENT)=0:SMALLEST(ELEMENT)=199
280     TEMPBIGGEST(ELEMENT)=0:TEMPSMALLEST(ELEMENT)=199
290 NEXT ELEMENT
300 XSTEP=2*(256/(XSIZE+1))
310 YSTEP=2*(60/(YSIZE+1))
320 FOR Y=0 TO YSIZE
330     XSCREENADJUST=6+YSTEP*Y
340     YSCREENADJUST=199-YSTEP*Y
350     FOR X=FIRSTX TO LASTX
360         Z=(HEIGHT*(MSCALE*ASC(INPUT$(1,#1))+BSCALE))
370         XSCREEN=XSCREENADJUST+XSTEP*X
380         YSCREEN=YSCREENADJUST-Z
390         IF TYPE=0 THEN GOTO 540
400         IF Y=0 THEN GOTO 530
410         DYSCREEN=YSCREEN-OLDLINE(X,2)
420         M=DYSCREEN/YSTEP
430         B=YSCREEN-M*XSCREEN
440         IF ABS(M)<1 THEN GOTO 500
450         IF DYSCREEN>0 THEN STEPI=1 ELSE STEPI=-1
460         FOR I=0 TO CINT(DYSCREEN) STEP STEPI
470             PY=CINT(OLDLINE(X,2)+I):PX=CINT((OLDLINE(X,2)+I-B)/M):GOSUB 830
480         NEXT I
490         GOTO 530
500         FOR I=1 TO YSTEP
510             PX=CINT(OLDLINE(X,1)+I):PY=CINT(M*(OLDLINE(X,1)+I)+B):GOSUB 830
520         NEXT I
```

```

530     OLDLINE(X,1)=XSCREEN:OLDLINE(X,2)=YSCREEN
540     IF X<>FIRSTX THEN GOTO 570
550         PX=CINT(XSCREEN):PY=CINT(YSCREEN):GOSUB 830
560         GOTO 690
570     DYSCREEN=YSCREEN-YOLDSCREEN
580     M=DYSCREEN/XSTEP
590     B=YSCREEN-M*XSCREEN
600     IF ABS(M)<1 THEN GOTO 660
610     IF DYSCREEN>0 THEN STEPI=1 ELSE STEPI=-1
620     FOR I=0 TO CINT(DYSCREEN) STEP STEPI
630         PY=CINT(YOLDSCREEN+I):PX=CINT((YOLDSCREEN+I-B)/M):GOSUB 830
640     NEXT I
650     GOTO 690
660     FOR I=1 TO XSTEP
670         PX=CINT(XOLDSCREEN+I):PY=CINT(M*(XOLDSCREEN+I)+B):GOSUB 830
680     NEXT I
690     XOLDSCREEN=XSCREEN:YOLDSCREEN=YSCREEN
700 NEXT X
710 FOR ELEMENT=1 TO 639
720     IF TEMPBIGGEST(ELEMENT)>BIGGEST(ELEMENT)
730         THEN BIGGEST(ELEMENT)=TEMPBIGGEST(ELEMENT)
740     IF TEMPSMALLEST(ELEMENT)<SMALLEST(ELEMENT)
750         THEN SMALLEST(ELEMENT)=TEMPSMALLEST(ELEMENT)
760     TEMPBIGGEST(ELEMENT)=0:TEMPSMALLEST(ELEMENT)=1000
770 NEXT ELEMENT
780 NEXT Y
790 CLOSE 1
800 BEEP
810 A$=INKEY$:IF A$="" THEN 790
820 KILL "improc#3.dat"
830 SYSTEM
840 END
850 IF (PY>BIGGEST(PX))OR(PY<SMALLEST(PX)) THEN PSET(PX,PY)
860 IF PY>TEMPBIGGEST(PX) THEN TEMPBIGGEST(PX)=PY
870 IF PY<TEMPSMALLEST(PX) THEN TEMPSMALLEST(PX)=PY
880 RETURN

```



## Appendix C.

### Hybrid computer-optical processing with inexpensive liquid crystal television

Matt Young and Matt Weppner

U.S. National Bureau of Standards, Electromagnetic Technology Division,  
325 Broadway, Boulder, Colorado 80303, USA

#### Abstract

We describe a computer-optical processing system that uses an inexpensive liquid crystal (LCD) television monitor and a selective holographic filter for coherent pattern recognition. Specifically, we use a digital computer to generate an edge enhanced image of an object, expose a Fourier transform hologram of this image, and use the hologram as a sort of matched filter for recognizing the original object in real time.

#### Introduction

It may not be true that great minds think alike, but it is demonstrably true that a handful of researchers have roughly simultaneously discovered a new liquid crystal, or LCD, television set for use in optical processing.<sup>1-5</sup> The value of such a device, of course, is that it is not self luminous (like a CRT) but, rather, can be used as a spatial light modulator for generating coherent images at video rates.

Typical LCD TVs cost less than US\$200 and may be addressed by a TV camera or by a micro-computer with a video frame digitizer. They therefore promise to allow coherent video images and computer generated patterns for holography and optical processing into almost any optics laboratory. We anticipate that they will find their greatest application in the input plane of the optical processor, where they are well suited, for example, to matched filtering and incoherent correlation experiments. Devices with higher resolution than the currently available values of about 120 x 140 pixels (vertical times horizontal) should additionally be useful for change or defect detection<sup>6</sup> and, to a limited degree, to computer generated holograms.<sup>5</sup> In applications where the transform plane can be magnified significantly, the monitor may be located there and used as a computer generated spatial filter or, possibly, a holographic filter.

The devices on the market today are designed for recreation and consist of a twisted nematic LCD screen glued between crossed polarizers. They are viewed in transmission and through a diffuser inclined at an angle of about 45° to the horizon. Removing the diffuser, modifying the hinge on the device, and building a rigid frame takes several hours.

The LCD panel we use is about 55 x 42 mm<sup>2</sup> and uses a raster formed by a grid of fine wires; the pixels are about 0.35 x 0.39 mm<sup>2</sup>.<sup>7</sup> The polarizers are not flat and cause a serious aberration in the transform plane<sup>1</sup> as well as a lack of space invariance in spatially coherent systems.<sup>4</sup> The aberration may be corrected by contacting optical glass plates with index matching fluid to both sides of the display or by removing the polarizers entirely and using external polarizers.

The fine wire raster causes many diffraction orders in the transform plane of a coherent processor; these orders contain the information about the raster itself. Therefore, for many applications, it will be necessary to use low pass filtering to eliminate all but the lowest diffraction order of the grid. For this purpose we use a two-stage system with two transform planes.<sup>1</sup> The low pass filtering is performed in the first stage; this leaves the transform plane of the second stage completely unobstructed so that filters or holographic plates may be easily located there.

The LCD display is also well suited to incoherent correlations, where the object is illuminated coherently but diffusely, a hologram is recorded, and correlations are performed with spatially incoherent illumination.<sup>8,9</sup> (In this case, the quality of the polarizers is irrelevant.) Such applications have been hindered in the past by the lack of a suitable incoherent display with the same wavelength as the argon laser; the LCD monitor allows the argon laser to be used throughout the experiment.

#### Hybrid optical processor

The optical system we use is an extension of that described in reference 1 and is shown here as Fig. 1. An unpolarized 3-mW He-Ne laser beam is split by an uncoated, wedged beam splitter; one of the reflected beams is used as the reference beam in a matched filtering experiment. The transmitted beam is spatially filtered with a 10x microscope objective and a

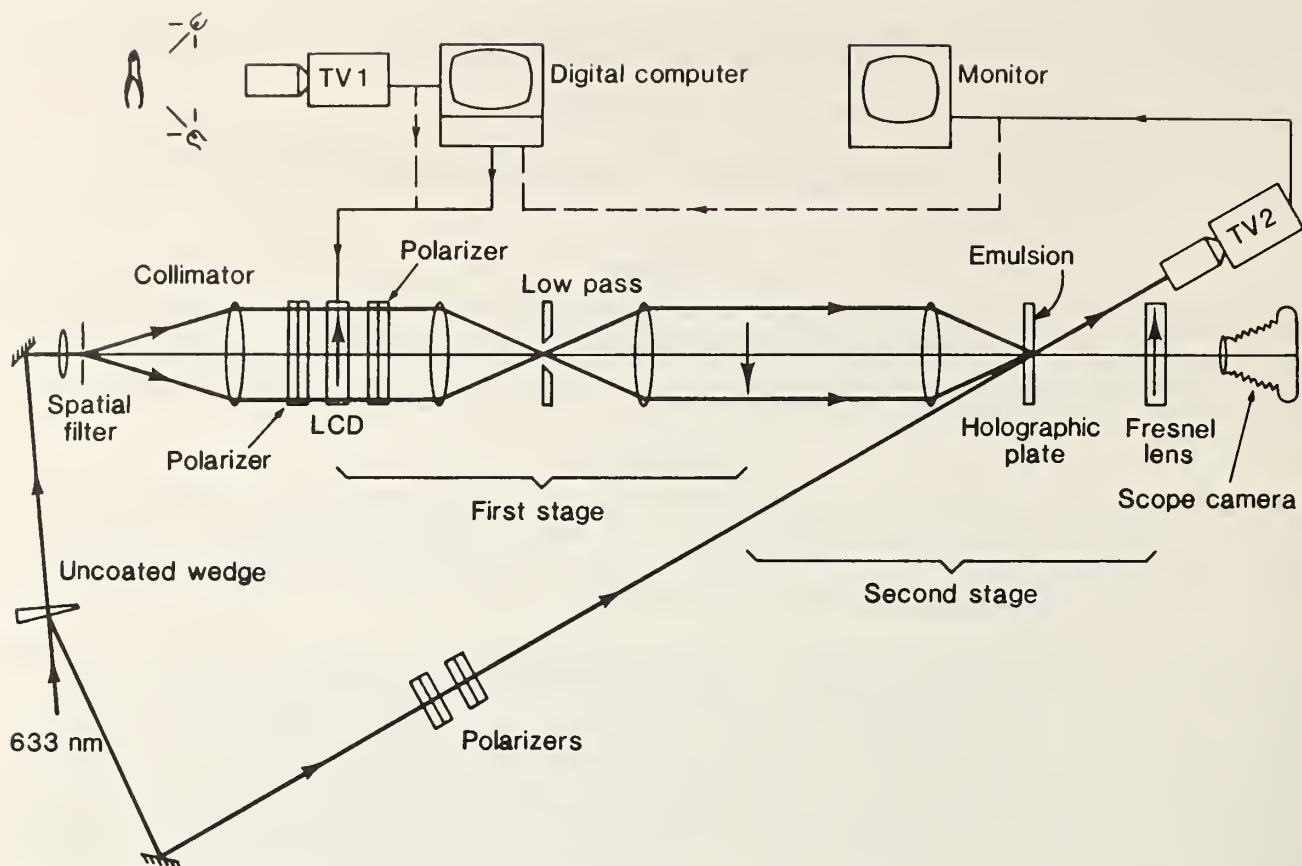


Figure 1. Hybrid processor consisting of a two-stage Fourier optical system that includes a liquid crystal, or LCD, TV monitor as input. A digital computer with a frame digitizer addresses the LCD monitor and a matched filter is recorded on the photographic plate. Camera TV2 displays the cross-correlation functions on a monitor or, alternatively, connects to the frame digitizer, as shown by the dashed line. The three Fourier transform lenses have 250-mm focal length,  $f/5.6$ .

40- $\mu\text{m}$  pinhole, collimated with a 360-mm lens, and directed into the two-stage processor. This system consists of a conventional 4-f processor followed by a single-lens processor in which the image is magnified slightly for ease of photographing. The lenses in the processor are all 210-mm copy or enlarging lenses. A Fresnel lens is inserted into the final image plane to serve as a field lens.

We located the LCD monitor in the object plane of the first stage. To optimize the system's performance, we peeled the polarizers from the screen and used external polarizers. We enclosed both the polarizers and the screen in liquid gates by contacting them on both sides to optical glass flats with an index matching oil. The oil has had no effect on the electrical performance of the monitor.

The monitor is addressed by a microcomputer with a video frame digitizing board. To evaluate the performance of the monitor, we generated some patterns, such as that shown in Fig. 2. This pattern is radially symmetric and runs linearly from black in the center to white at the edge, or, in terms of the frame digitizer, from 0 at the center to 255 at the edge. (We used this pattern because the monitor gave unpredictable results with uniform gray or with a gray scale whose gradient was either vertical or horizontal. We attribute this to a defect in the video analog-to-digital (A-to-D) converter, but we do not know whether or not it is just an idiosyncrasy of our sample. A newer model seems to give somewhat better results in this regard.) After optimizing the polarizers and the monitor's brightness control and A-to-D converters, we used a silicon detector to scan along a diameter of the radial transmittance pattern. The result is shown as Fig. 3A. The monitor has an overall contrast ratio of only 10 to 1 and cannot respond to the full dynamic range of the TV camera or frame digitizer. We attribute the low contrast in part to the presence of the wires; these mainly add unwanted light to the nominally black areas, probably because of diffraction or some perturbation of the liquid crystals.



Figure 2. Radial transmittance function displayed on the LCD monitor, A, before low pass filtering, B, after low pass filtering. Contrast ratio of B is 5 times that of A.

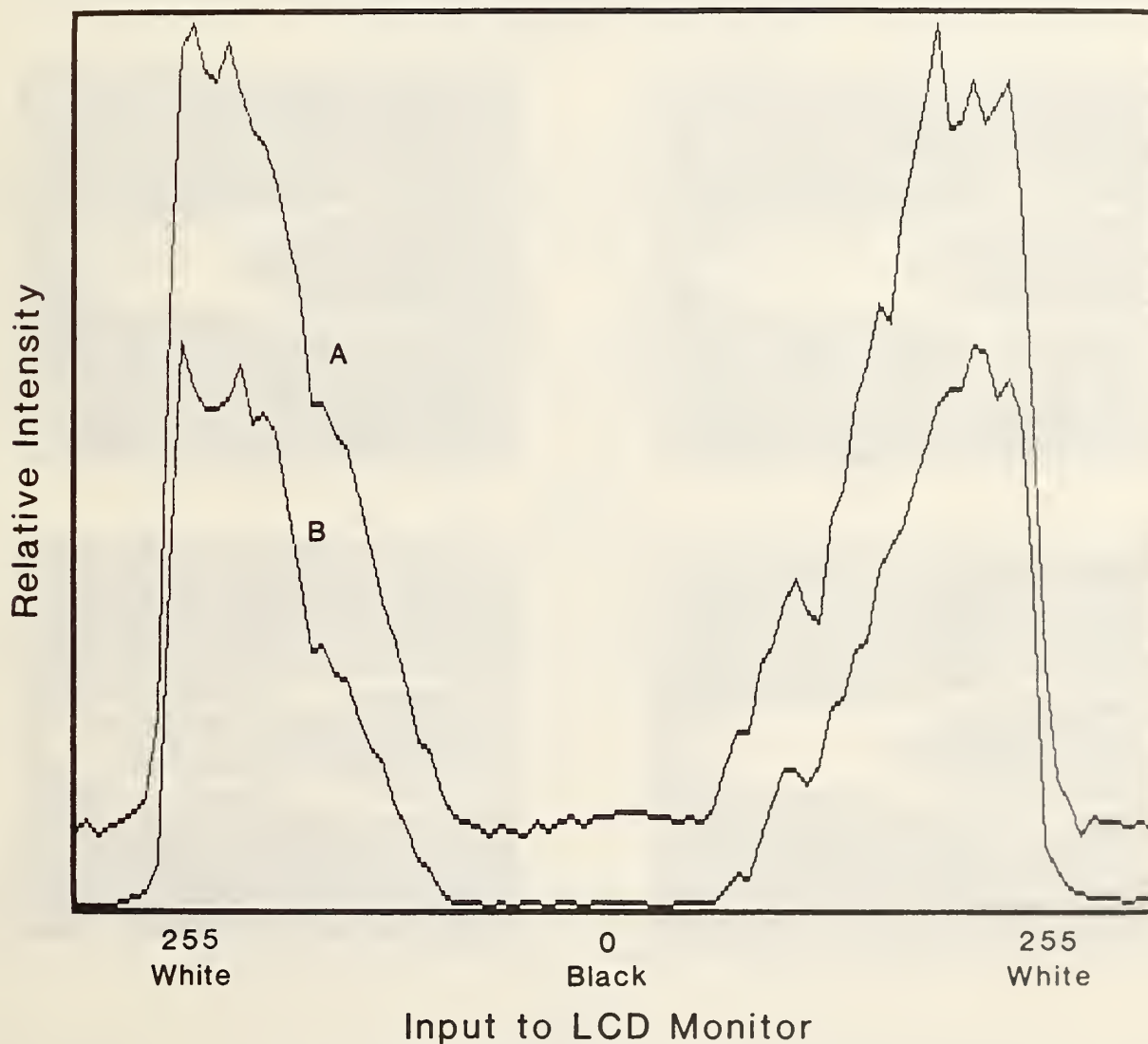


Figure 3. Transmittance as a function of position along a diameter of each of the screens shown in Fig. 2. Horizontal scale is the output of the frame digitizer on a range from 0 (black) to 255 (white).

When we located a 0.4-mm low pass filter in the transform plane of the first stage, we thereby eliminated the wire grid from the image (Fig. 2B) and measured a contrast ratio (Fig. 3B) of about 50 to 1. The monitor, however, still does not respond to the full dynamic range of the incoming TV signal, as is shown by the flat portions of the curve near the center and the edges of the display. These signify a relative lack of both shadow and highlight detail; adjusting the brightness or the video A-to-D converter level simply shifts the sloped portions of the curve horizontally toward or away from the center of the graph.

To complete the system, we located a holographic plate in the transform plane of the second stage and introduced the reference beam through a pair of polarizers for intensity control. (The reference beam angle of about  $15^\circ$  was determined by the size of the transform lens.) For these experiments, we did not collimate the reference beam, so it had a waist at the location of the laser, about 2 m from the hologram. This may reduce the position invariance of the system, but we did not check for position invariance.

#### Experiment

For our experiment, we illuminated a pair of pliers diffusely with two incandescent lamps and white, matte paper diffusers. Using a TV camera fitted with a zoom lens, we captured and stored an image of the pliers with the computer and the frame digitizer. The image displayed on the LCD monitor is shown in Fig. 4A prior to low pass filtering with the optical system. Figure 4B shows the same image after spatially filtering with the 0.4-mm aperture to eliminate the wire grid. When the low pass filter is chosen properly, only the grid is eliminated from the picture; there is no loss of resolution of the image itself.

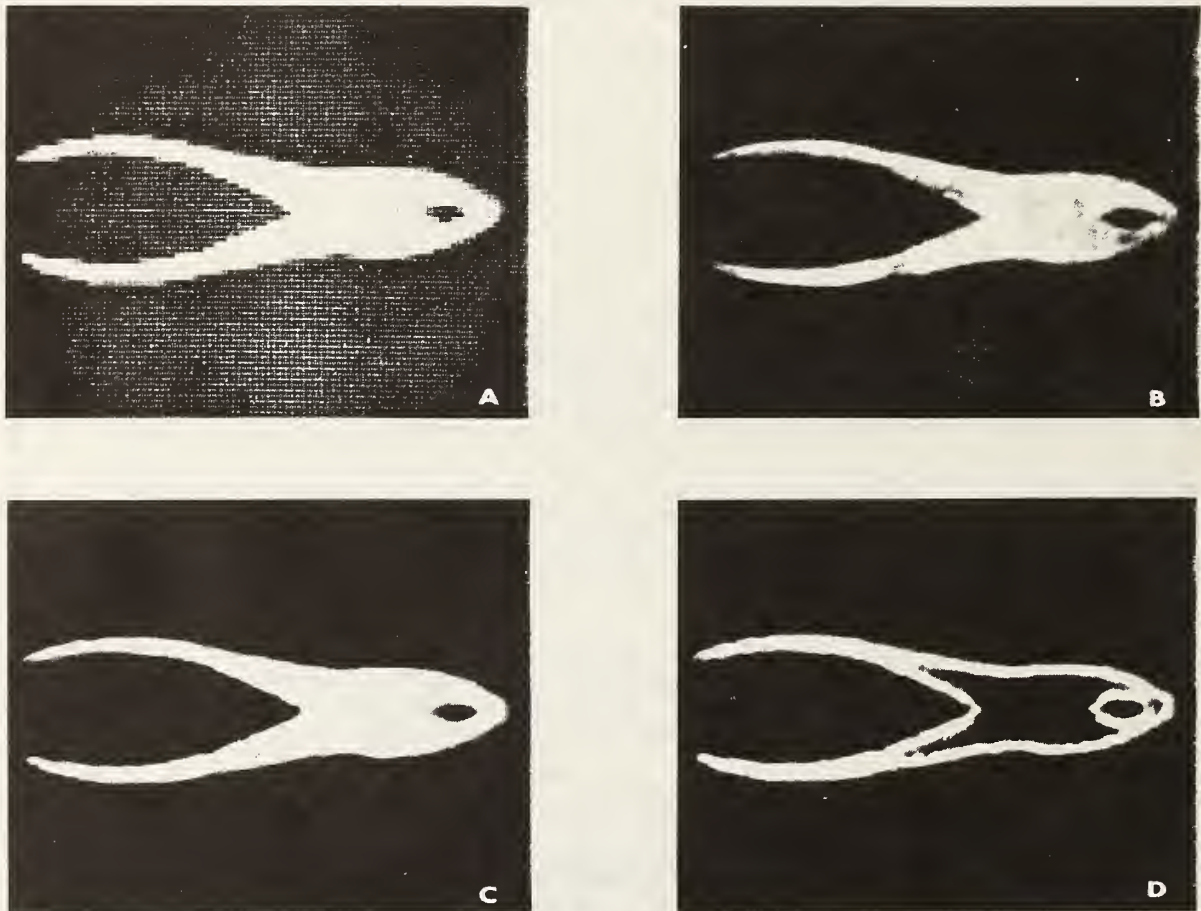
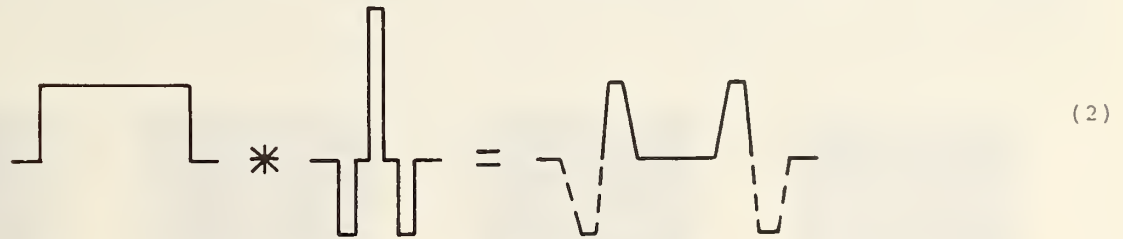


Figure 4. Pair of pliers used as a subject for pattern recognition experiment and displayed on the LCD monitor. A, image of the pliers themselves. B, low pass filtered to remove the pixels on the LCD screen. C, binary image of the pliers, wherein values below a threshold are set equal to 0 and values above that threshold are set equal to 255. D, edge enhanced model of the pliers.

Next we used the computer to prepare a clipped, or binary, image of the pliers, Fig. 4C, where intensities above a certain level are set equal to 255 (white) and those below that level are set equal to 0 (black). In the one dimensional analogy,

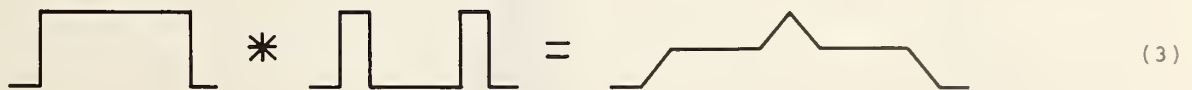


Next, we performed a two dimensional edge enhancement in which we retained only the inside edge of the binary image; that is, the image is black except for a white region that runs along the inside edge of the binary image. Symbolically,

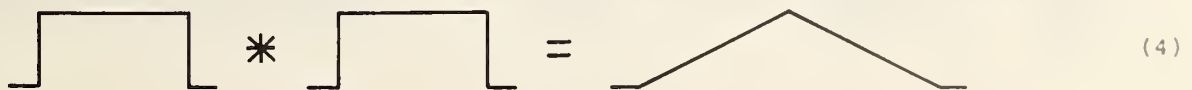


where the asterisk (\*) denotes cross correlation and values less than zero are interpreted as zero. We refer to the resulting image as the model; it is shown in Fig. 4D. The width of the enhanced edge is determined by the number of zeros between the peaks of the kernel of the cross correlation integral. We chose this width to be two pixels on the LCD monitor; this corresponds to about six pixels on a conventional monitor with the full resolution of about 380 pixels.

We used the model, somewhat in the manner of references 8 and 9, as the object in a pattern recognition experiment. We inserted a holographic plate in the second transform plane and recorded a Fourier transform hologram of the model. The value of using the model, rather than the object itself, is that the cross correlation function of the model and the object,



is, with an irregular two dimensional object, sharper than the autocorrelation function of the object itself,



and permits greater discrimination against similar objects. In addition, the model is located on a black background, and its Fourier transform has a relatively weak dc or zero-order component. As a result, the exposure of the photographic plate is comparatively uniform and allows a hologram to be recorded with roughly constant diffraction efficiency for all relevant spatial frequencies.

### Results

Placing the emulsion side away from the transform lens, we exposed some Fourier transform holograms to serve as matched filters for the model. We processed some of them in a potassium ferricyanide bleach (15 mg/mL for 4 min). These may be expected to have a diffraction efficiency perhaps 10 times that of amplitude transmission holograms and are much more tolerant of variations of exposure.<sup>10</sup> Our best filter was just such a phase hologram.

Using a second TV camera denoted TV2 and shown in Fig. 1, we viewed the autocorrelation function of the model on a monitor. Because the image is mostly black, we switched the camera's automatic gain control off to avoid overexposing the autocorrelation spot. The AGC was left off during all the observations that follow. We did not need to synchronize the camera TV2 to the computer, evidently because the decay time of the LCD panel was longer than the frame rate of 1/60 s.

Figure 5A shows a photograph of the central portion of the monitor. The spot is sharp and has a width of about six TV lines; this agrees well with the expected value of two pixels on the lower resolution LCD monitor. The spot can also be seen clearly with the eye and is surrounded by a diffraction ring that does not appear in the photograph.

Next, we displayed the clipped image of the object on the LCD monitor, without moving the apparatus or adjusting either TV2 or the brightness of the monitor. The resulting cross correlation function is shown in Fig. 5B. We then switched to real time and displayed the real object on the LCD monitor. The relative brightness of the object was substantially less than the 255, or white, of the computer processed images. The cross correlation with

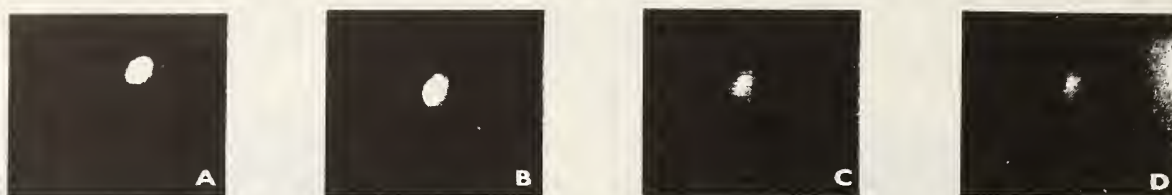


Figure 5. Correlation functions as seen by TV2 displayed on a monitor. A, autocorrelation function of the model. B-D, cross correlation function of the model with (B) clipped tool, (C) tool in real time, and (D) tool rotated 2°. All photographs were taken with the same exposure.

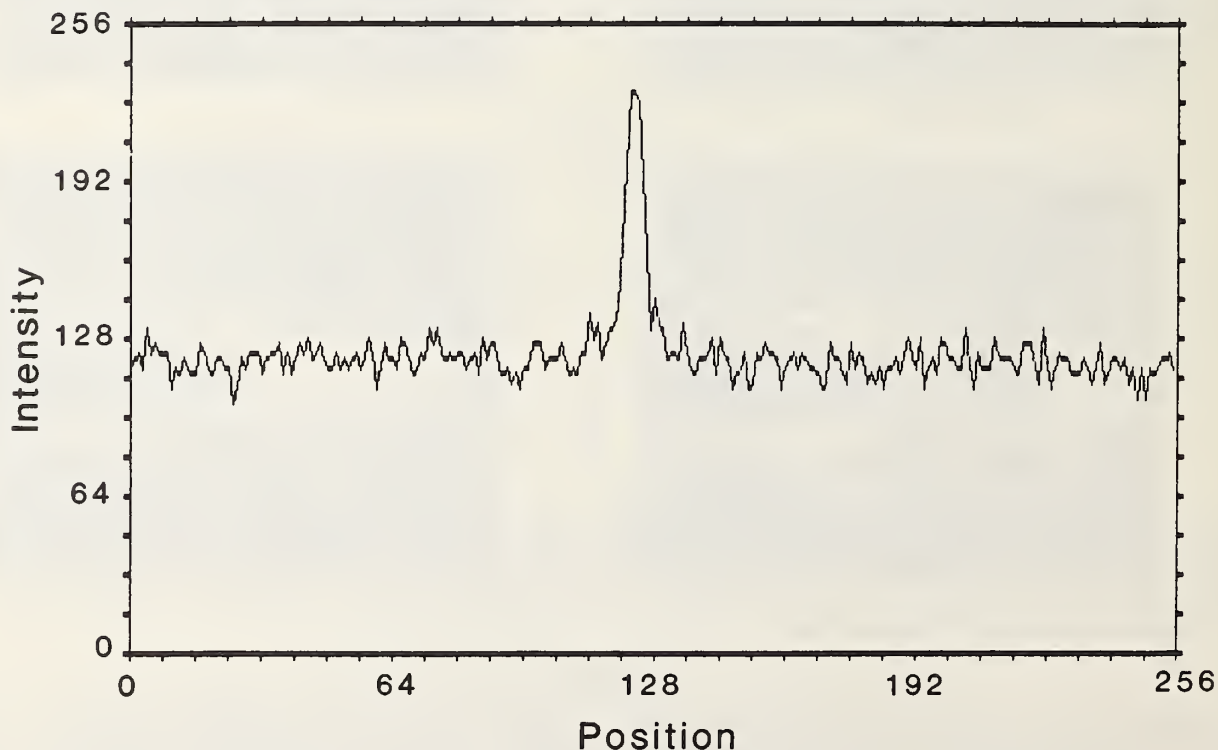


Figure 6. Intensity as a function of position along a horizontal TV line through the center of the cross correlation peak of Fig. 5C. Average of four frames.

the model was weaker but plainly visible on the monitor (Fig. 5C). Finally, we rotated the object 2°. The cross correlation function in this case was still visible and is shown in Fig. 5D.

We then switched the camera, TV2, to the computer and digitized its output. We could not record the autocorrelation function of the model, because that would have required two frame digitizers; indeed, this is one of the reasons that we designed our experiment to examine the cross correlation of the unprocessed object, rather than a processed image, with the model.

We have made no attempt to optimize the diffraction efficiency of the hologram. What was plainly visible on the monitor was not so plain to the computer; the eye does a marvelous job of smoothing and averaging. We therefore digitized four frames and averaged them. Figure 6 is a graph of the relative intensity across one horizontal line in the averaged picture, multiplied by 3 for clarity of display. The constant value of about 115 is simply dark current from the TV camera.

Figure 7 shows a histogram of the entire screen (of which Fig. 6 is a single line). (All values greater than zero are shown as small ticks, whether or not they properly round to zero.) The highest value, 228, represents the cross correlation peak. The average value of the dark current is about 118, and the dark noise fluctuations range from roughly 104 to 132. If we take this to be  $\pm 3$  sigma, we estimate the signal-to-noise ratio (SNR) to be approximately 24.

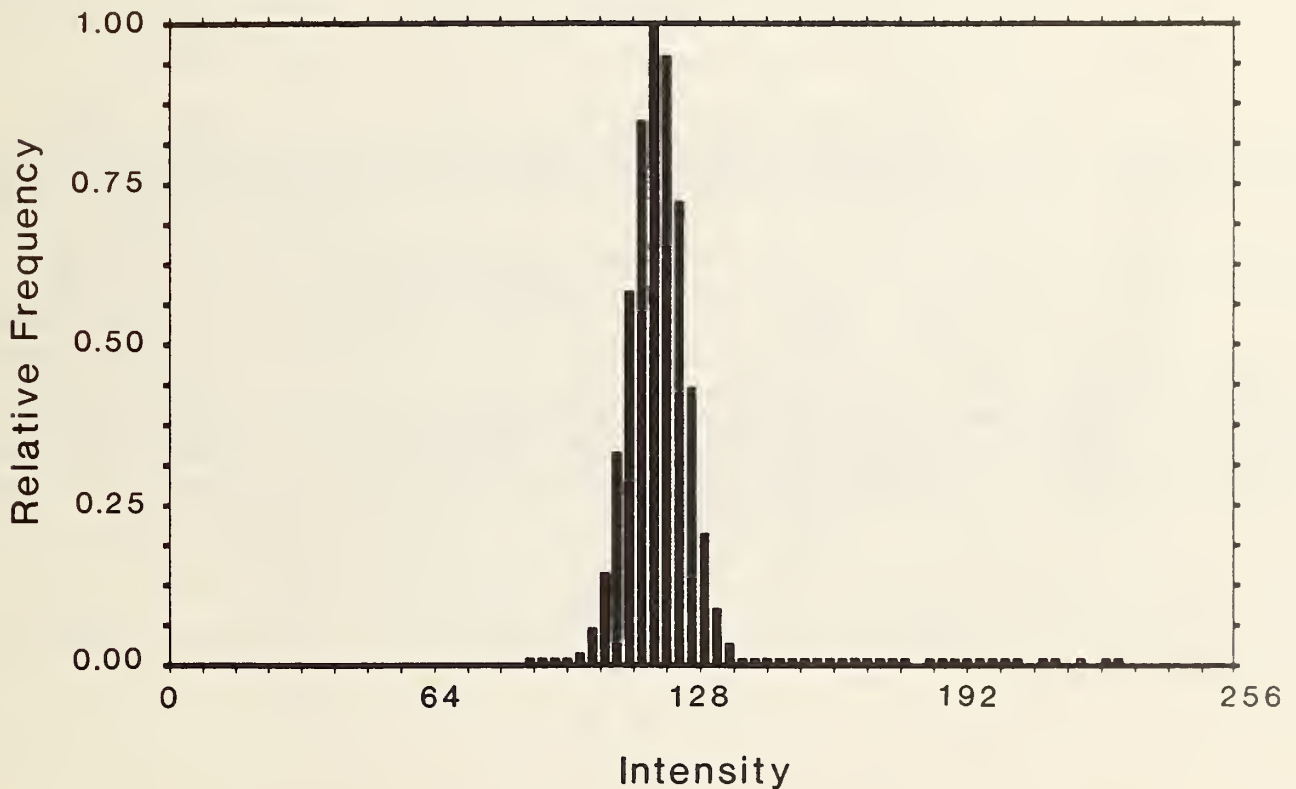


Figure 7. Histogram of the entire screen from which Fig. 6 is derived. The signal peak is 228, the noise average 118, and  $\pm 3$  sigma about 28. The signal-to-noise ratio is about 24.

#### Discussion

We have used a digital computer to generate an edge enhanced (but non-negative) model of a tool, prepared a Fourier transform hologram of the model, and cross correlated the model in real time with an image of the original tool.

We might have achieved better discrimination if we had either edge enhanced the tool in real time (as in reference 8) or edge enhanced with positive and negative values (as in

reference 9). For practical reasons, we did neither. The latter, in particular, would have required an encoding scheme that required a space-bandwidth product greater than the LCD monitor offered and would not have achieved significantly different results unless we had used the equivalent of heterodyne detection. We could have achieved a similar result by edge enhancing optically (with a bandpass filter in the first transform plane); this achieves a phase shift in the electric field at the location of an edge (as is shown by the presence of a zero of intensity at the edge<sup>11</sup>). We chose, however, to edge enhance by computer since we thought that this would be more readily controllable and result in a simpler, singly peaked intensity distribution at the edge.

Finally, we have performed our experiment using coherent processing. This is far more efficient than incoherent processing, which requires diffused illumination, and has allowed us to use a small He-Ne laser rather than the much higher power argon laser used in references 8 and 9. Except for the limited space-bandwidth product offered by the LCD monitor, most other considerations are about equal, and the use of the small and inexpensive He-Ne laser may therefore militate in favor of coherent processing for this sort of pattern recognition.

#### Acknowledgments

Matt Weppner is also with the Electrical Engineering Department, University of Colorado, Boulder, Colorado. We are indebted to Edie DeWeese for her excellent manuscript preparation and to Chuck Petersen of EPIX, Inc. for his help with image processing algorithms. Contribution of the National Bureau of Standards, not subject to copyright.

#### References

1. Young, M., "Low-Cost LCD Video Display for Optical Processing," Appl. Opt., Vol. 25, pp. 1024-1026. 1986.
2. Gregory, D. A., "Real-time Pattern Recognition Using a Modified Liquid Crystal Television in a Coherent Optical Correlator," Appl. Opt., Vol. 25, pp. 467-469. 1986.
3. McEwan, J. A., Fisher, A. D., Rolsma, P. B., and Lee, J. N., "Optical-Processing Characteristics of a Low-Cost Liquid-Crystal Display Device," J. Opt. Soc. Am. A, Vol. 2, No. 13, p. P8. 1985. Abstract only.
4. Liu, H.-K., Davis, J. A., and Lilly, R. A., "Optical-Data-Processing Properties of a Liquid-Crystal Television Spatial Light Modulator," Opt. Lett., Vol. 10, pp. 635-637. 1985.
5. Tai, A. M., "Low-Cost LCD Spatial Light Modulator with High Optical Quality," Appl. Opt., Vol. 25, pp. 1380-1382. 1986.
6. Watkins, L. S., "Inspection of Integrated Circuit Photomasks with Intensity Spatial Filters," Proc. IEEE, Vol. 57, pp. 1634-1639. 1969.
7. Radio Shack Catalog Number 16-151. The use of trade names is intended to specify experimental procedure precisely and does not imply endorsement by the National Bureau of Standards. Other manufacturers' products may work as well or better.
8. Sherman, R. C., Grieser, D., Gamble, F. T., Verber, C. M., and Dolash, T., "Hybrid Incoherent Pattern Recognition System," Appl. Opt., Vol. 22, pp. 3579-3582. 1983.
9. Katzir, Y., Young, M., and Glaser, I., "Pattern Recognition Using Incoherent OTF Synthesis and Edge Enhancement," Appl. Opt., Vol. 24, pp. 863-867. 1985.
10. Young, M. and Kittredge, F. H., "Amplitude and Phase Holograms Exposed on Agfa-Gevaert 10E75 Plates," Appl. Opt., Vol. 8, pp. 2353-2354. 1969.
11. Young, M., "Linewidth Measurement by High-Pass Filtering: A New Look," Appl. Opt., Vol. 22, pp. 2022-2025. 1983.

U.S. DEPT. OF COMM. <b>BIBLIOGRAPHIC DATA SHEET</b> (See instructions)		1. PUBLICATION OR REPORT NO. NBSIR 87-3065	2. Performing Organ. Report No.	3. Publication Date April 1987
4. TITLE AND SUBTITLE IMAGE PROCESSING FOR OPTICAL ENGINEERING APPLICATIONS				
5. AUTHOR(S) Matthew B. Weppner and Matt Young				
6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)  NATIONAL BUREAU OF STANDARDS DEPARTMENT OF COMMERCE WASHINGTON, D.C. 20234			7. Contract/Grant No.	
			8. Type of Report & Period Covered	
9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)				
10. SUPPLEMENTARY NOTES  <input type="checkbox"/> Document describes a computer program; SF-185, FIPS Software Summary, is attached.				
11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)  This Internal Report describes the development and testing of image processing software designed for optical engineering applications. Image processing functions in this software include two-dimensional Fourier transforms, convolution, noise reduction, multiple image resolutions, and low-level image processing functions. The software also contains image information display tools including Gaussian beam and g-profile characterization for optical fiber measurements. The necessary image file input/output routines are presented in the software and are used to read and store images in conjunction with other image processing software, digitizing cameras, and output display devices.				
12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons) computer processing; Fourier optics; g-profile; Gaussian beam; image processing; images; optical fiber characterization; optical processing; single mode fiber.				
13. AVAILABILITY  <input checked="" type="checkbox"/> Unlimited <input type="checkbox"/> For Official Distribution. Do Not Release to NTIS <input type="checkbox"/> Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.  <input checked="" type="checkbox"/> Order From National Technical Information Service (NTIS), Springfield, VA. 22161			14. NO. OF PRINTED PAGES  110	
			15. Price	









